推薦論文

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†

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 1 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，フィルタリング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 種類の手法を提案し，ルール集合の圧縮を行う．また，実際に運用されている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，70%前後までルール数を削減することができた．

A Packet Filtering Rules Compression by Decomposing into Matrixes

Katsushi Matsuda†

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 deletion, verbose rules revision and rules combination, and implemented a prototype system. The experiment using actual rule sets showed that our system could reduce the number of rules by about 30 percent.

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 1 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十〜数百，ときには数千に及ぶ．しかも各ルールの条件は複数の属性の組合せになるため，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のように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いか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わらないルールが多数存在する．また，作成者や作成時間が異なるために生じた不必要に複数に分割されたルール等も多数存在する．これら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．

本稿では，パケットフィルタリングのルール集合を

† NEC サービスプラットフォーム研究所
Service Platforms Laboratories, NEC

本論文の内容は 2006 年 3 月のコンピュータセキュリティ研究会にて報告され，CSEC 研究会主査により情報処理学会論文誌への掲載が推薦された論文である．

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削除する方法および複数のルールを統合してルール数を削減する方法について述べる．

2. 関連研究

パケットフィルタリングルールの分析で最も基本的な研究は，packet classification という分野である．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ものである（文献 1), 2) に詳しい）．計算幾何学のデータ構造とアルゴリズムを用いることで速度とメモリ使用量のトレードオフに挑んでいる．

packet classification と類似したパケットフィルタリングルールの分析の成果として，関連するルールを抽出するクエリーベースのツールがある[3),4)]．これらのツールは，あるルール集合によって結局どのようなアクセス制御状態が実現されるのかを求めることができるという意味で，ネットワーク管理者にとって有益である．

本稿で述べる不要なルールを検出する先行研究には，文献 4)〜6) 等がある．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で，優先順位の高いルールによって隠されたルール（潜伏ルールと呼ぶ）を検出することが可能である[4)]．Jalili らは高レベルな管理ポリシではあるが，優先順位の低いルールによって代替できる冗長なルール（冗長ルールと呼ぶ）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間の関係を分析しているため，不要なルールを完全に検出できているとはいえない．すなわち，ルールの一部が冗長であり，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，不要なルールではなく，複数のルールを 1 つのルールとすることで結果的にルール集合のルール数を削減する研究を筆者らはまだ見つけていない．

3. マトリックス分解

パケットフィルタリングルールを分析する場合，一般的には条件範囲の関連性をルール単位で比較することが多い．実際前章であげた先行研究では 1 対 1 のルールの比較でのみ分析を行っている．しかし，多対 1 や多対多の比較でないと不要なルールと判断できない場合も多い．

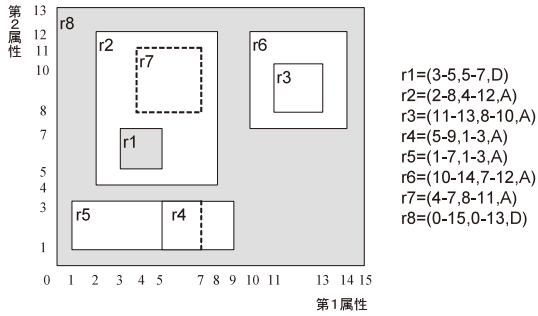

図 1 ルール集合の例
Fig. 1 An example rule set.

本稿で報告するルール分析方法は，ルール単位ではなく，マトリックスと呼ぶ極小多次元立方体単位を用いている．本章では，ルール集合をマトリックス空間データと呼ぶ表現形式に変換す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で範囲指定されたすべ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．このマトリックスを用いて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．以下，マトリックス分解について詳細に述べる．

3.1 マトリックス生成

まずはルール集合からマトリックスを生成する．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（送信元アドレス，送信元ポート，送信先アドレス，送信先ポート，プロトコル）が用いられているが，ここでは簡単のため，2 種類の属性で説明する．

8 個のルールを持つルール集合の例とそれを 2 次元平面で表現したものを**図 1** に示す．図の X 軸は第 1 属性，Y 軸は第 2 属性であり，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 次元平面図ではそれぞれ白色と灰色で表す．

マトリックス生成は，まず条件属性ごと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，重複をなくしたうえでソートする．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（マトリックス）を作る．図 1 のルール集合の例では，第 1 属性の境界点が 14 個，第 2 属性の境界点が 11 個あるため，130 個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，属性が 2 個の場合は，(第 1 属性の範囲，第 2 属性の範囲，ルールリスト）となる．なお，属性が n 個の場合のマトリックス m は，$m = (v_1, v_2, \cdots, v_n, R_m)$ となる．ここで，v_i は第 i 属性の範囲，R_m は次節で述べるルールとマトリックスの対応付けで行われるマトリックス m を要素として持つルールのリストである．

3.2 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を要素として持つルールのリストを対応付ける．この対応付けによって，ルールの書式には末尾にマトリックスリストが付与される．すなわち，属性が n 個の場合のルール r_i は，$r_i = (v_1, v_2, \cdots, v_n, a, M_{ri})$ となる．ここで，a はルールのアクションであり，M_{ri} は r_i を構成するマトリックスのリストである．

ルール r_i のマトリックスリスト M_{ri} の順序には意味がないが，マトリックス m のルールリスト r_m の順序は，ルール集合の優先順位に準拠して並んでいる．たとえば，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，さらにその下に r8 があるため，r1 を構成するマトリックスのルールリストは，$\{\mathrm{r1}, \mathrm{r2}, \mathrm{r8}\}$ となる．

3.3 マトリックス空間データ

マトリックスとルール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．たとえば，分析の例として packet classification がある．マトリックス空間データの中からパケット属性に合致するマトリックスを参照することで packet classification が実現できる．

3.1 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．ルールの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[7]．

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．ルールの条件範囲の開始点と終了点に重複がないルール集合（ルール数 n 個，条件属性 k 種）の場合，$(2n-1)^k$ 個のマトリックスが生成されることになる．ただし，この生成数は原理上の最大値であり，実際に用いられているルール集合で生成されるマトリックスは $(3n/2)^{k/2}$ 個程度である．この増加を抑制するために，不要なマトリックスを削除する．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．これによって，平均 74.8%のマトリックスが削減できることを確認している[7]．

4. ルール圧縮

本章では，フィルタリングルールの不要なルールを削除する方法および複数のルールを統合してルール数を削減する方法，すなわちルール圧縮について述べる．ルール圧縮は前章で述べたマトリックス空間データを用いることで効果的に実現できる．ルール数を削減することによって，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し，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．

4.1 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 2 種類の方法がある．1 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．冗長条件ルールの修正は直接ルールを削除するわけではないが，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，まずルール削除を行い，その後にルール統合を行う．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) 不要ルール再削除
(iv) ルール統合

4.2 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，優先順位の高いルールによって隠されたルールであり，他方は，優先順位の低いルールによって代替できる冗長なルールである．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．

$$\text{Consealed}(r_i)\ \text{iff}$$
$$\forall m \left\{ (m \in M_{r_i}) \subseteq \sum_{j=1}^{i-1} r_j \right\} \quad (1)$$

$$\text{Verbose}(r_i)\ \text{iff}$$
$$\forall m\ (m \in M_{r_i})\{\text{act}(r_i) = \text{act}(\text{next}(m, r_i))\} \quad (2)$$

ここでルール集合を $R = \{r_1, r_2, \ldots, r_n\}$，マトリックスを m とする．なお，M_{ri} は，ルール r_i のマトリックスリストである．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 R_m 中のルール r_i の次のルールを next(m, r_i) とする．たとえば 3.2 節の例（{r1, r2, r8}）の場合，next(m, r2) = r8 となる．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すべ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すべ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，式 (1) および式 (2) を組み合わせた以下の式 (3) が必要十分条件となる．式 (3) により，あるルールの半分が冗長ルールの条件を，残りの半分が潜伏ルールの条件を満たすような場合であっても不要ルールとして削除することが可能となる．

$$\text{Removable}(r_i)\ \text{iff}$$
$$\forall m\ (m \in M_{r_i}) \left(m \subseteq \sum_{j=1}^{i-1} r_j \right)$$
$$\vee \big(\text{act}(r_i) = \text{act}(\text{next}(m, r_i))\big) \quad (3)$$

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，ルール r_i が不要ルールかどうかの判定は，最大 $|r_i|$ 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．

4.3 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すべ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．

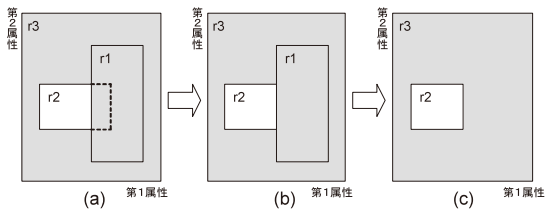

図 2 不要ルールの新規発生
Fig. 2 An unnecessary rule occurs.

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．さら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．

$$\text{Revisable}(r_i)\ \text{iff}$$
$$\forall m\,(m \in M)\ \exists M\,(M \subset M_{r_i})\ \exists k\,\forall l\,(k, l \in K)\big\{$$
$$\text{range}(M_{r_i}, k) \neq \text{range}(M, k)$$
$$\wedge\ \text{range}(M_{r_i}, l) = \text{range}(M, l)$$
$$\wedge \big(r_i \neq \text{top}(m)\ \vee\ \text{act}(r_i) = \text{act}(\text{next}(m, r_i))\big)\big\} \quad (4)$$

ここで，M は多次元立方体となるマトリックス集合，K は条件属性の全集合とする．range(M, k) は，多次元立方体 M のある次元（属性条件 k）の最小値と最大値からなる範囲である．マトリックス m のルールリスト R_m 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すべてが 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，たかだか $2k$ 回のマトリックス集合の生成と調査で可能である．

4.4 不要ルール再削除

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．これ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)）．

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，何度でも繰り返すことが可能である．

4.5 ルール統合

ルール統合とは，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，ある 2 個のルールが統合可能かどうかを判定するフェーズと，その統合し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からなる．

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できる．

$$\begin{aligned}&\mathrm{Unifiable}(r_i, r_j)\ \text{ iff}\\ &\quad \forall \nu\ (\nu \in (r_i + r_j))\big\{\mathrm{revised}(r_i) \wedge \mathrm{revised}(r_j)\\ &\quad \wedge\ \mathrm{num}(r_i + r_j) = \mathrm{num}(r_i) + \mathrm{num}(r_j)\\ &\quad \wedge\ (\mathrm{in}(r_i, \nu) \vee \mathrm{in}(r_j, \nu))\big\} \qquad (5)\end{aligned}$$

ここで ν は多次元立方体の頂点，ルール r_i が冗長条件ルールでないことを $\mathrm{revised}(r_i)$，ルール r_i の構成マトリックスの数を $\mathrm{num}(r_i)$，ルール r_i の条件範囲の中に頂点 ν が含まれていることを $\mathrm{in}(r_i, \nu)$ 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すべ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否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よい．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．

統合し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 通りがある．なお，図 3 では軸を表示していないが，図 1 および図 2 と同様に X 軸方向が第 1 属性で Y 軸方向が第 2 属性であるとする．(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．

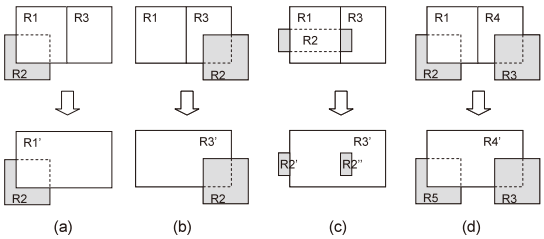

図 3 統合ルールの挿入箇所
Fig. 3 Insert point of unified rules.

(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よい．(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．

5. フィルタリングルール分析システム

マトリックス分解を用い，ルール圧縮を行うフィルタリングルール分析システムを実装した．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，筆者らが開発した汎用ポリシ記述言語 SCCML を用いた[8]．SCCML は，パケットフィルタリング等のアクセス制御系のセキュリティ機器および侵入検知システム等の監視系のセキュリティ機器の動作をモデル化し，標準化団体 OASIS が制定したインターネット経由の情報アクセスに関する制御ポリシを表現する XACML（eXtensible Access Control Markup Language）[9] を拡張した XML 形式の言語である．

実装したシステムの動作例を示す．**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と，実際の運用に近いルール集合となっている．

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．図 4 のルール集合を SCCML に変換した後，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．

```
     ! 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
Fig. 4 An experimental rule set.

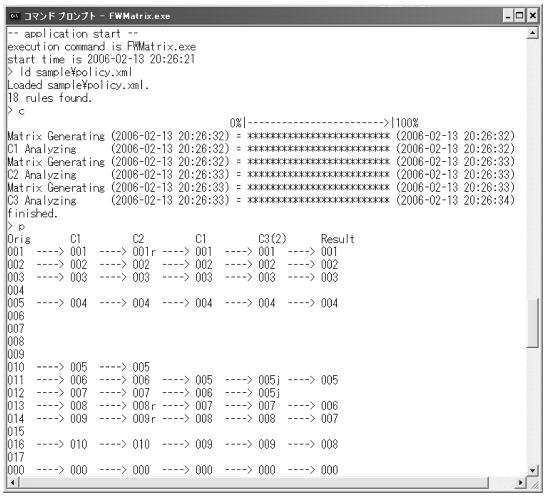

図 5 システムの実行例
Fig. 5 An execution example of the system.

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不要ルール再削除を行う 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 が統合されている．結局，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%）．

6. 実験結果

実際に用いられているルール集合を用いた圧縮率の測定を行った．その結果を**表 1** に示す．ルール集合 A は，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であり，ルール集合の作成・維持は研究者がボランティアで行っていたものである．残りのルール集合は，フィルタリングルールの管理サービスを行っている部門で実際にサービスしているルール集合である．

表 1 ルール圧縮の測定結果
Table 1 Experimental results of rule compression.

ルール集合	元ルール数	不要ルール数	統合ルール数	最終ルール数	圧縮率
A	525	54	123	348	66.3%
B	294	85	0	209	71.1%
C	31	0	0	31	100%

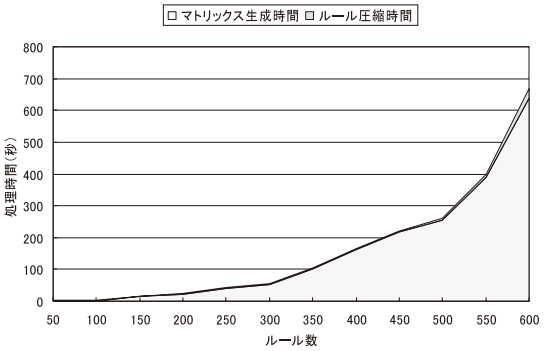

図 6 処理時間の測定結果
Fig. 6 An experimental result of run time.

表 1 は，各ルール集合が不要ルール削除，不要ルール再削除，ルール統合によって何個削除されたかを示している．冗長条件ルール修正に関しては，修正されたルール数を示している．

測定結果から，専門家がメンテナンスを行うルール集合（B および C）でさえ，ルール数が多くなる（B）と，不要なルールが多数存在していることが判明した．非専門家がメンテナンスを行っているルール集合（A）の場合，ルール統合の効果が大きい．理由は，あるサービスに関して複数の利用者が個別にアクセス許可の申請を行っており，そうして作成されたルールが統合されたためである．なお，ルール集合 A のルール統合の処理は 2 回行われた．

圧縮後のルール数は元のルール数の 70%前後にまで削減できる．本システムを用いることで，パケットフィルタリングシステムの管理コストを低減することができる．

また，処理時間の測定を行った．その結果を**図 6** に示す．処理時間の測定で用いた計算機は，CPU が Pentium 4 3.4 GHz で搭載メモリが 2 G バイトのパーソナルコンピュータである．3.3 節で述べたように，実

際に利用されているルール集合で平均的に生成されるマトリックス数は $(3n/2)^{k/2}$ 個程度である．このマトリックス数を近似するルール集合自動生成ツールを作成した．

ツールを用いて，50〜600 個の 50 個刻みのルール数を持つルール集合を 3 セット生成し，処理時間を測定した．システムの処理は，マトリックス生成，不要ルール削除，マトリックス生成，冗長条件ルール修正および不要ルール再削除，マトリックス生成，ルール統合の順に行われる．図 6 では，マトリックス生成時間の合計および各手法によるルール圧縮時間の合計の平均値を示している．マトリックスは，ルール集合で作る多次元空間を各ルールの各条件属性で範囲指定されたすべ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ため，ルール数が増加するとマトリックス数は指数的に増加する．マトリックス生成時間はこれらのマトリックスのメモリ確保に要する時間であるため，図 6 のようにルール数の増加に従い，指数的に処理時間が増している．一方，ルール圧縮時間は図 6 ではほぼ一定のように見えるが，実際にはルール数の増加にともないルール圧縮時間も指数的に増加している．ルール圧縮時間のうち，不要ルール削除と冗長条件ルール修正は，式 (3) および式 (4) に示すように，ルール数とほぼ比例した処理時間を要し，ルール統合は，式 (5) に示すように，ルール数の 2 乗に比例した処理時間を要する．しかし，いずれの処理もマトリックス空間データ中のマトリックスを参照する程度の処理で収まるため，比較的短時間で完了している．

ルール数が 650 個前後になると，生成されるマトリックスが必要とするメモリ量は 2 G バイトを超える．32 ビット OS の Windows XP でアプリケーションが扱える最大メモリ空間は 2 G バイトであるため，本システムでルール圧縮が可能なルール集合のルール数の上限は約 600 個となる．圧縮率の測定で用いたルール集合 A のルール数は 525 個で約 200 台の端末を接続しているネットワークで実際に用いているルール集合であるが，メモリ不足に陥ることなくルール圧縮を行うことができた．ルール集合 A の端末数とルール数の関係が一般的に成り立つと仮定すると，本システムで対応可能なネットワークの規模は約 200 台となる．

7. おわりに

パケットフィルタリングのルール集合を分析することを可能にするマトリックス分解手法と，それを用いた不要ルール削除とルール統合によるルール圧縮方法について述べた．マトリックスは，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しており，各種圧縮方法を定式化するのに適している．

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ルール数を 70%前後まで削減することができることが確認できた．パケットフィルタリングのルール数は少ないほど，ネットワーク管理者の管理コストは低くなり，その結果としてセキュリティも向上することが見込まれる．すなわち，本システムによってセキュリティの向上が可能となる．

提案したルール統合方式は，最適なルール統合にならない可能性があることが判明している．2 個のルールを順に統合しているため，どのルールから統合するかによって最終的に統合されるルールの数が異なるためである．統合されるルール数が最多になるまですべてのルールの組合せを試すことで最適なルール統合が可能となる．

参考文献

1) Feldmann, A. and Muthukrishnan, S.S.: Tradeoffs for Packet Classification, *Proc. INFOCOM(3)*, pp.1193–1202 (2000).
2) Gupta, P. and McKeown, N.: Algorithms for Packet Classification, *IEEE Network*, Vol.15, No.2, pp.24–32 (2001).
3) Mayer, A., et al.: Fang: A Firewall Analysis Engine, *Proc. 2000 IEEE Symposium on Security and Privacy*, pp.177–187 (2000).
4) Eronen, P. and Zitting, J.: An Expert System for Analyzing Firewall Rules, *Proc. 6th Nordic Workshop on Secure IT-Systems* (*NordSec2001*), pp.100–107 (2001).
5) Jalili, R. and Rezvani, M.: Specification and Verification of Security Policies in Firewalls, *EuroAsia-ICT 2002*, LNCS 2510, pp.154–163 (2002).
6) Al-Shaer, E.S. and Hamed, H.H.: Modeling and Management of Firewall Policies, *IEEE Trans. Network and Service Management*, Vol.1-1 (2004).
7) 松田勝志：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).
8) 岡城純考，松田勝志，小川隆一：セキュリティ運用管理のためのポリシー言語 SCCML，情報処理学会研究報告，Vol.2004, No.129, pp.89–94 (2004).
9) OASIS: XACML Version1.1.

http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf

(平成 18 年 7 月 21 日受付)
(平成 19 年 7 月 3 日採録)

推 薦 文

ファイヤウォールにおけるパケットフィルタリングのルールの冗長さに着目し，マトリックス分解と呼ぶ提案方式によりルール数の削減を試みている．ルールの統廃合の方法に新規性があり，一連の研究の中で創意工夫を重ねることで，約 70%の圧縮を実現している．ルータにとっても管理者にとってもコストを削減できる点，XML に基づく一般的な中間形式を導入し，複数のルータベンダーの形式に対応している点で実用性も高い．

（コンピュータセキュリティ研究会主査 寺田真敏）

松田 勝志（正会員）

1967 年生．1994 年大阪大学大学院基礎工学研究科博士後期課程修了．同年 NEC 入社．現在，NEC サービスプラットフォーム研究所主任研究員．1997 年から奈良先端科学技術大学院大学連携講座准教授を兼務．1999 年〜2000 年米国ワシントン大学に Visiting Scholar として滞在．1989 年度 AI 学会全国大会優秀論文賞，1999 年，2001 年情報処理学会全国大会優秀賞受賞．博士（工学）．WWW 検索技術，システムセキュリティ技術，情報分析技術等の研究に従事．人工知能学会，IEEE Computer Society 各会員．