Run-Based Trie から構成される決定木の枝刈り法

原田崇司[1]　田中賢[1]　三河賢治[2]

[1] 神奈川大学大学院理学研究科情報科学専攻

[2] 新潟大学学術情報基盤機構情報基盤センター

2015 年 11 月 6 日

目次

パケットフィルタリングとは

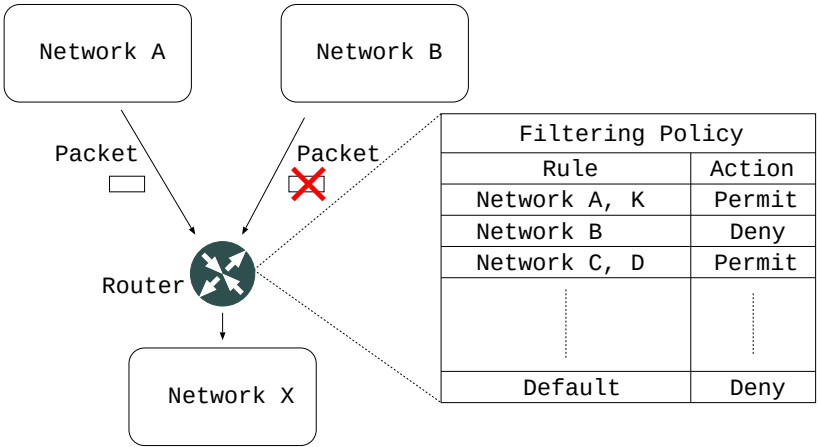


Figure : 入ってくるパケットをポリシーに従ってルータで分類

パケットフィルタリングとは

パケットフィルタリングを行う関数 f

$$f : \{ 0, 1 \}^{dW} \rightarrow \{ P, D \}$$

パケットフィルタリングとは，長さ dW の 0,1 のビット列に対して P, D の値を割り当てる関数

フィルタリングルールの形式

i番目のルール

$$R_i^v = b_1 b_2 \dots b_{dW} \quad (b_i \in \{0, 1, *\},\ v \in \{P, D\})$$

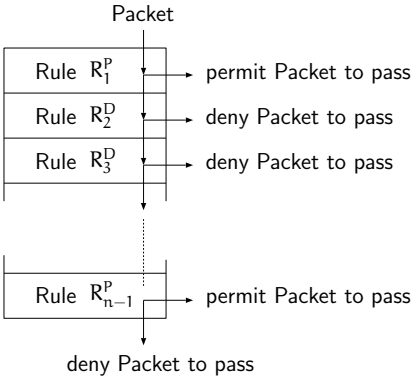
Table : フィールドが一つの例

Filter	F_1
R_1	$0\ *\ 1\ *$
R_2	$*\ 0\ 0\ 0$
R_3	$1\ 1\ 0\ 1$
R_4	$1\ 1\ *\ *$
R_5	$0\ *\ *\ 1$
R_6	$*\ 1\ *\ *$

Table : フィールドが二つの例

Filter	F_1	F_2
R_1	$*\ 1\ *\ 0$	$*\ 0\ 0\ *$
R_2	$1\ *\ 0\ 0$	$1\ 1\ *\ 1$
R_3	$*\ 1\ 1\ *$	$0\ 1\ 1\ 1$
R_4	$0\ 1\ *\ *$	$*\ 1\ *\ 1$
R_5	$1\ 0\ 1\ 1$	$0\ 0\ 1\ *$
R_6	$*\ 0\ 1\ 0$	$1\ 1\ 0\ 0$

線型探索によるパケットフィルタリングの実装



if Packet match R_1 then permit Packet else

if Packet match R_2 then deny else

if Packet match R_3 then deny else

if

if Packet match R_{n-1} then permit else

deny Packet

Figure : 上位のルールから順番にパケットとルールをひとつひとつ比較

パケットは最後のデフォルトルール R_n^D とは比較されない.

パケットフィルタリングに対する理論的な研究

- フィルタリングルールリスト最適化
  - SubGraph Merging（Tapdiya *et al* [1] 2009 年）
  - 昆金による方法（昆金ら [2] 2013 年）
  - 竹山による方法（竹山ら [3] 2013 年）
- 到着パケットに合致する最優先ルールを高速に見つける
  - 階層型トライ（Srinivassan *et al* [4] 1998 年）
  - HiCuts, HyperCuts（Gupta *et al* [5], Singh *et al* [6] 2000, 2003 年）
  - Grouper（Ligatti *et al* [7] 2010 年）
  - Run-Based Trie（決定木を用いる）（三河ら [8] 2015 年）

何故 Run-Based Trie

HiCuts，HyperCuts，階層型トライは，任意のビットマスクを含むルール ($0*11*00*10*$ の様な歯抜けのルール) に対応しない

任意のビットマスクを含むルールに対応するデータ構造である Run-Based Trie を提案

どうして決定木

ルールは一旦追加されると危険性の有無を確認できない

追加したルールは消せないのでルール数は増える一方

ルール数が増えると探索時間が長くなる

ルール数に依存しない探索法である決定木を用いた探索法を提案
(前掲の方法, Run-Based Trie の素朴な探索法はルール数に依存)

Run-Based Trie

Table : 任意のビットマスクを含むルールのルールリスト

Filter	F_1
R_1	$*0*1$
R_2	0000
R_3	$0*00$
R_4	$0*1*$
R_5	1100
R_6	$*01*$

Filter	F_1
R_7	$**10$
R_8	$01**$
R_9	$*11*$
R_{10}	$*000$
R_{11}	$*1*1$
R_{12}	$***1$

上記のルールリストのルール中に存在する連の例：

R_1 の 2 ビット目からの 0，4 ビット目からの 1
R_3 の 1 ビット目からの 0，3 ビット目からの 00

Run-Based Trie

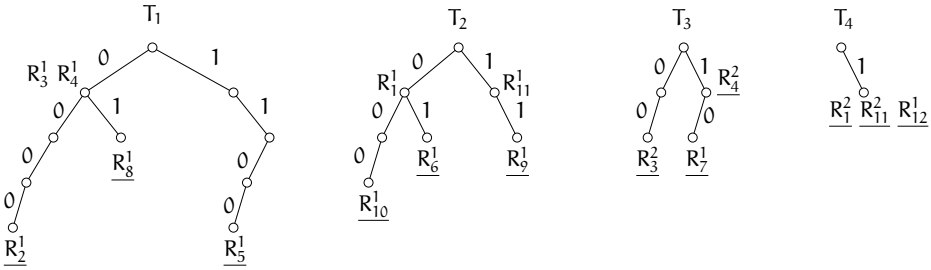


Figure : Run-Based Trie

ルールリスト中の各連の開始位置 i ビット毎にトライ T_i を構成

ルールを構成する連の中で最後の連には下線を引く

$$R_1 = *0*1, \quad R_3 = 0*00$$

Simple Search

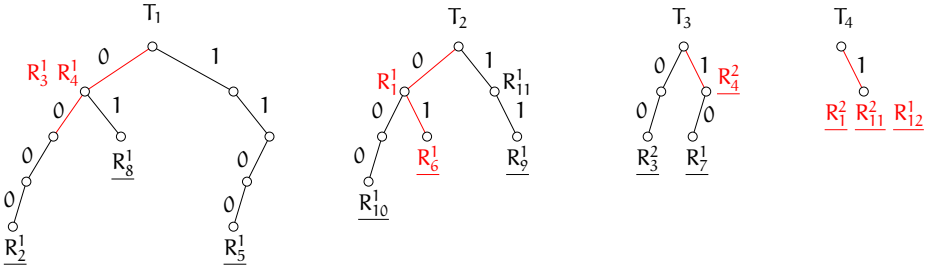


Figure : パケット 0011 で Run-Based Trie を辿る

R_6, R_4, R_1, R_{12} に合致．この中で優先度の一番高い R_1 を返す．

探索時間計算量：$O(NdW + (dW)^2)$　　（ルール数 N に依存）

集合族

各トライ T_i を辿った際に得られるパケットと連との合致パタンの集合を S_i で表す.

$$S_1 = \{ \{R_3^1, R_4^1\}, \{\underline{R_2^1}, R_3^1, R_4^1\}, \{R_3^1, R_4^1, \underline{R_8^1}\}, \{\underline{R_5^1}\}, \phi \}$$
$$S_2 = \{ \{R_1^1\}, \{R_1^1, \underline{R_{10}^1}\}, \{R_1^1, \underline{R_6^1}\}, \{R_{11}^1\}, \{\underline{R_9^1}, R_{11}^1\}, \phi \}$$
$$S_3 = \{ \{\underline{R_3^2}\}, \{\underline{R_4^2}\}, \{\underline{R_4^2}, R_7^1\}, \phi \}$$
$$S_4 = \{ \{\underline{R_1^2}, \underline{R_{11}^2}, R_{12}^1\}, \phi \}$$

決定木

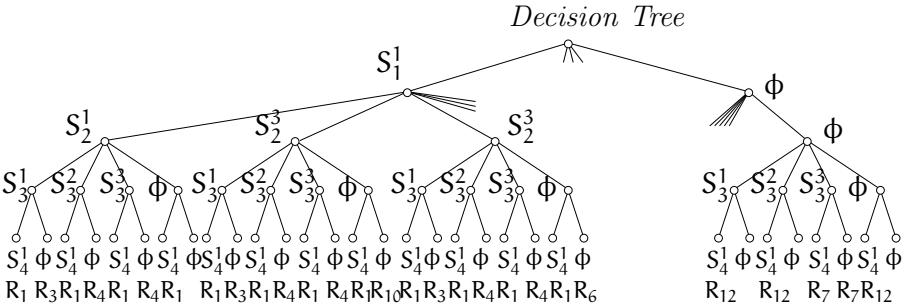


Figure : 決定木

Run-Based Trie を用いて決定木を降りるだけで探索終了

探索時間計算量：$O((dW)^2)$　　（ルール数 N に依存しない）

決定木はメモリを食いすぎ

探索時間は $O((dW)^2)$ だが空間計算量が $O(N^{dW})$

⇩

空間計算量が膨大で全く実用的でない

⇩

決定木の枝刈りアルゴリズムが必要

「$T_1, T_2, \ldots, T_k$ での合致の仕方から T_{k+1} での合致の仕方としてあり得ないものを排除」という方針で枝刈りを実行

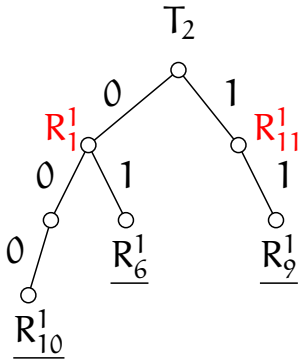
集合族の元の表記

$$
\begin{aligned}
S_1 &= \{\ \{R^1_3, R^1_4\},\ \{\underline{R^1_2}, R^1_3, R^1_4\},\ \{R^1_3, R^1_4, \underline{R^1_8}\},\ \{\underline{R^1_5}\},\ \phi\ \} \\
S_2 &= \{\ \{R^1_1\},\ \{R^1_1, \underline{R^1_{10}}\},\ \{R^1_1, \underline{R^1_6}\},\ \{R^1_{11}\},\ \{\underline{R^1_9}, R^1_{11}\},\ \phi\ \} \\
S_3 &= \{\ \{\underline{R^2_3}\},\ \{\underline{R^2_4}\},\ \{\underline{R^2_4}, R^1_7\},\ \phi\ \} \\
S_4 &= \{\ \{\underline{R^2_1}, \underline{R^2_{11}}, R^1_{12}\},\ \phi\ \}
\end{aligned}
$$

集合族の元を，その元を得るパケットのビットパタンに置き換える．

$$
\begin{aligned}
S_1 &= \{\ 0,\ 0000,\ 01,\ 1100,\ \phi\ \} \\
S_2 &= \{\ 0,\ 000,\ 01,\ 1,\ 11,\ \phi\ \} \\
S_3 &= \{\ 00,\ 1,\ 10,\ \phi\ \} \\
S_4 &= \{\ 1,\ \phi\ \}
\end{aligned}
$$

集合族内の不要な元の消去



S_2 の元 ϕ は，T_2 のトライを辿る際にいずれの連にも合致しないことを意味するが，$\{R^1_1\}$ (0) と $\{R^1_{11}\}$ (1) の連が存在するので，いずれの連にも合致しないということはない．

S_2 から ϕ を削除

Figure : トライ T_2

$$S_2 = \{\ \{R^1_1\},\ \{R^1_1, \underline{R^1_{10}}\},\ \{R^1_1, \underline{R^1_6}\},\ \{R^1_{11}\},\ \{\underline{R^1_9}, R^1_{11}\},\ \phi\ \}$$

親のビット列と比較

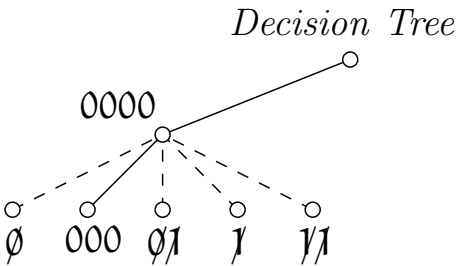


Figure : 親のビット列と比較

上位のトライ (T_1) を 0000 と辿るパケットは，現在のトライ (T_2) を

- 0 としか辿れないということはない
- 01 とは辿らない
- 1 とは辿らない
- 11 とは辿らない

⇩

0000 の子は 000 のみ

親の兄弟のビット列との比較による枝刈り

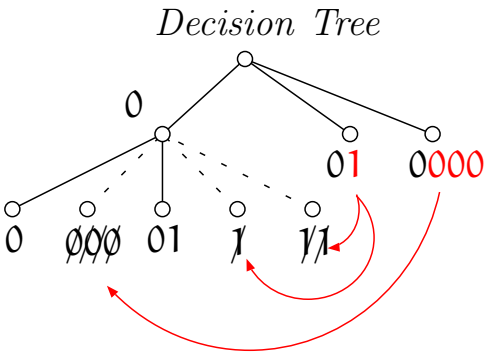


Figure : 親の兄弟ノードとの比較

パケットが T_1 で 0 のパタンに合致することは T_1 を

- 01 と辿れなかった
- 0000 と辿れなかった

ことを意味するので，そのようなパケットは T_2 を 1 とは辿らない．また 000 とも辿らない．

0 の子ノードとして 1 と 000 から始まるものを消去

子孫に枝刈り情報を伝える

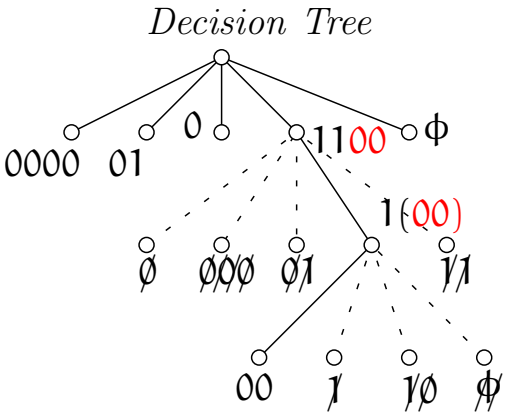


Figure : 子孫に枝刈り情報を伝える（祖父母のレベルの枝刈り情報が必要）

枝刈り法を適用した決定木

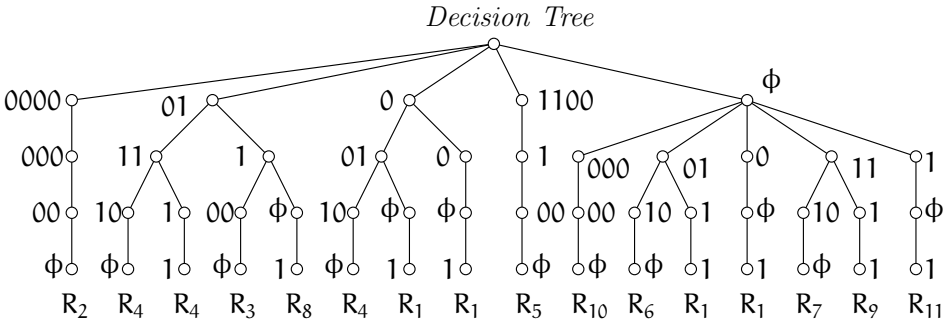


Figure : あり得ない辺を取り除いた決定木

ノード数: $396 \rightarrow 48$

枝刈り法を適用した決定木

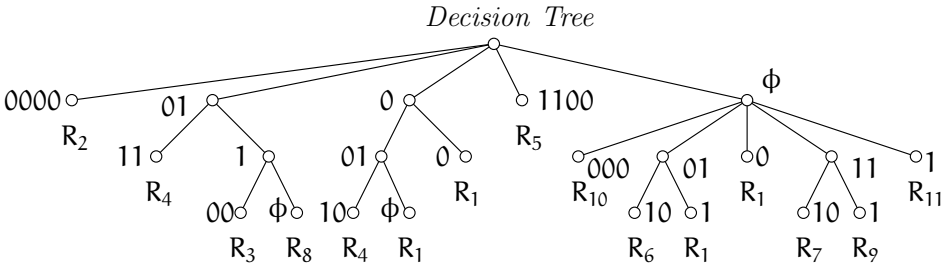


Figure : 枝分かれしないパスを短縮した決定木

ノード数: $48 \rightarrow 23$

実験環境

OS	:	CentOS Release 6.2
CPU	:	Intel Core i7-980X 3.33 GHz
主記憶容量	:	24GB
実装言語	:	C++
コンパイラ	:	gcc version 4.4.6

実験結果（100 ルール）

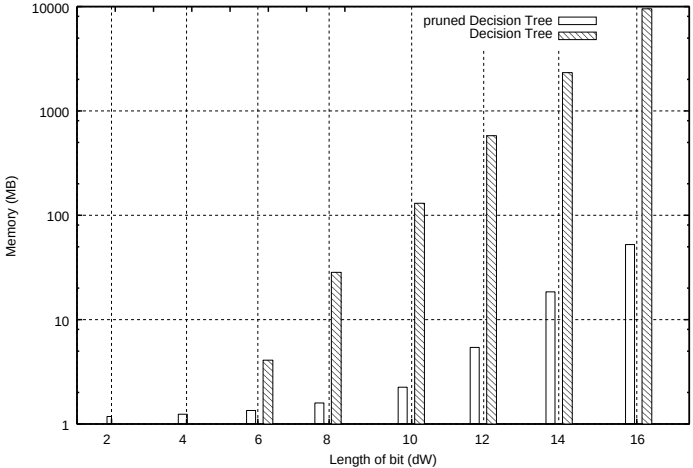


Figure : 決定木の構築に要するメモリ

実験結果（100 ルール）

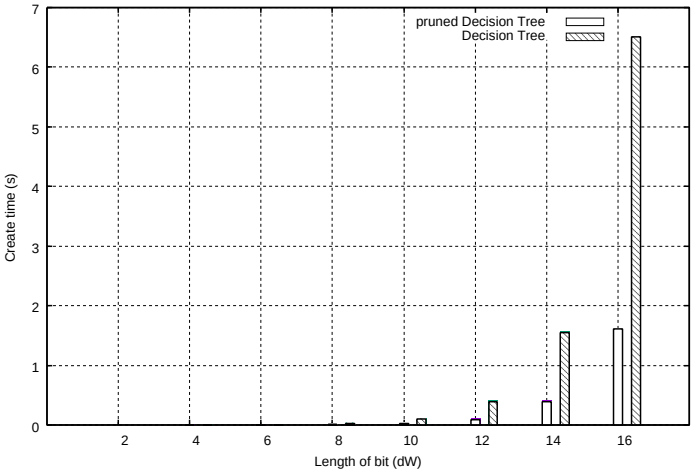
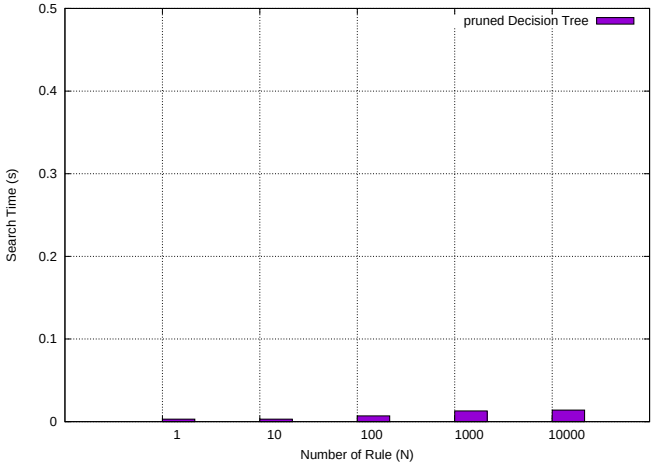


Figure : 決定木の構築に要する時間

実験結果（16ビット）



Figure： 決定木探索の探索時間

まとめと今後の課題

まとめ

- Run-Based Trie を探索するのに用いる決定木の空間計算量が $O(N^{dW})$ なので決定木の枝刈り法を提案
- 提案した枝刈り法の有効性を実装実験により 16 ビットまで確認

今後の課題

- 枝刈り法の時間，空間計算量の算出
- 提案した枝刈り法が無駄のない決定木を構成することの証明
- より長いビット長での枝刈り法の有効性の確認

- A. Tapdiya, and E. Fulp, "Towards optimal firewall rule ordering utilizing directed acyclical graphs," Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on, pp.1-6, Aug 2009.
- K. TANAKA, K. MIKAWA, and M. HIKIN, "A heuristic algorithm for reconstructing a packet filter with dependent rules," IEICE Trans. Commun., vol.96, no.1, pp.155-162, Jan 2013.
- K. Tanaka, K. Mikawa, and K. Takeyama, "Optimization of packet filter with maintenance of rule dependencies," IEICE Communications Express, vol.2, no.2, pp.80-85, Feb 2013.
- V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," SIGCOMM Comput. Commun. Rev., vol.28, no.4, pp.191–202, Oct. 1998.
- P. Gupta, and N. McKeown, "Classifying packets with hierarchical intelligent cuttings," Micro, IEEE, vol.20, no.1, pp.34-41, Jan 2000.
- S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp.213–224, New York, NY, USA, 2003, ACM.
- J. Ligatti, J. Kuhn, and C. Gage, "A packet-classification algorithm for arbitrary bitmask rules, with automatic time-space tradeoffs," Proceedings of the International Conference on Computer Communication Networks (ICCCN), pp.145–150, Aug. 2010.
- K. MIKAWA, and K. TANAKA, "Run-based trie involving the structure of arbitrary bitmask rules," IEICE Transactions on Information and Systems, vol.E98.D, no.6, pp.1206-1212, 2015.