ソフトウェアOpenFlowスイッチにおける Bitmask対応の高速なフロー検索手法の検討

NTT未来ねっと研究所
○日比智也，中島佳宏，高橋宏和，尾花和昭
2014/04/11

アジェンダ

- 背景
- 目的
- アルゴリズム検討
  - ハッシュによるアプローチ
  - トライ木によるアプローチ
- トライ木の実装手法検討
  - Double Array + vEB
  - LOUDUS
- 評価
- まとめ

背景

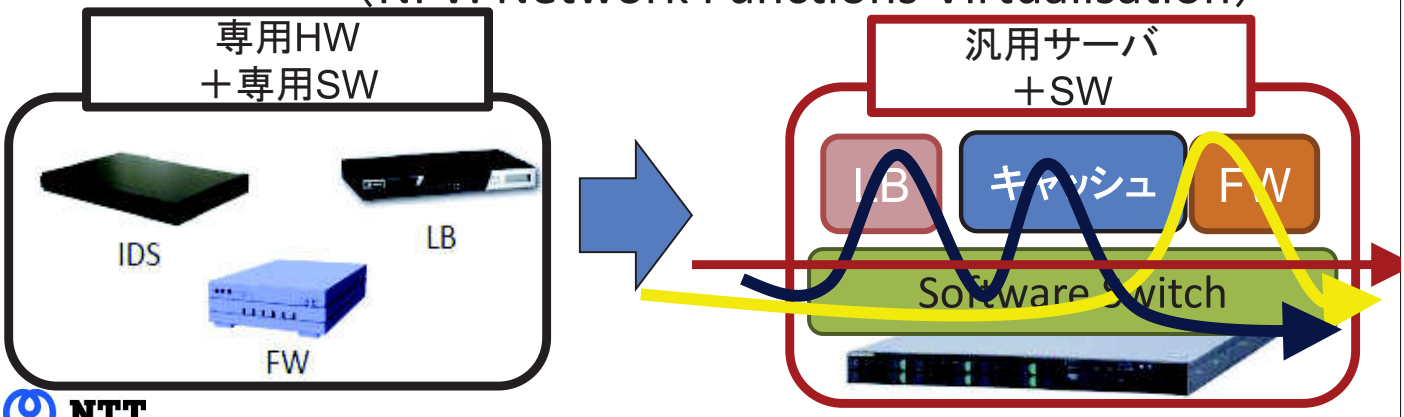
SDNやNFVを実現する基盤としてソフトウェアOpenFlowスイッチの高速化が求められている.

動向: 仮想化技術, クラウドの発展

CPU・バス・NIC等サーバの性能向上

トレンド: 専用HW&SW → 汎用HW&SW

(NFV: Network Functions Virtualisation)



NTT

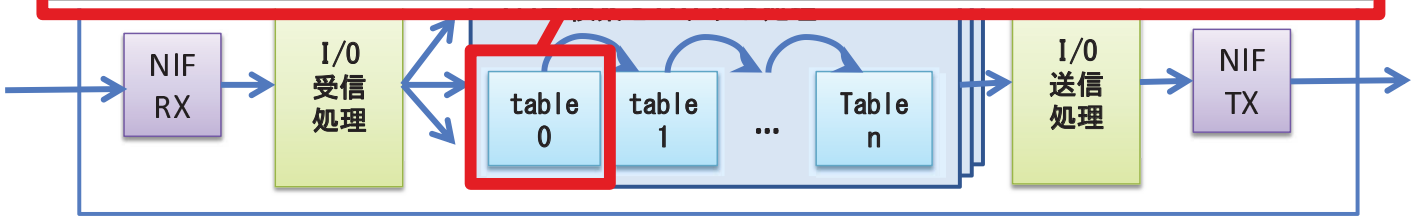
目的

OpenFlowのフロー検索はBitmaskを含む為, 検索の高速化が難しい. Bitmaskに対応した高速なフロー検索手法を検討・実装し, ソフトウェアOpenFlowスイッチの高速化を行う.

Bitmask: 特定のbitのみをLookupに使用
例) Src MAC = XX:XX:XX:11:11:XX

Wildcard:
特定のFieldはLookupに使用しない

Priority	MAC src	IPv4 dst	IPv4 src	L4 port dst	Instruction
100	00:00:00:11:11:00 /00:00:00:FF:FF:00	192.168.0.0 /255.255.0.0	192.168.1.0 /255.255.255.0	*	Apply-Actions
9999	*	192.168.0.1 /255.255.255.255	*	5533	Drop
0	*	*	*	*	Drop



NTT

アプローチと目標

アプローチ　既存アルゴリズムの活用
＋ 実装の工夫

目標性能
エントリ数　：　1M Entry
Lookup性能：　10M Lookup/sec
エントリ修正：　5分以内

必要機能　Bitmask対応

Bitmask対応の既存アルゴリズム

- **リファレンス実装(Hash + 線形サーチ)**
  - NIC Offload [Tanyingyong et al. 2011]
  - GPU Offload [Han et al. 2010]

  → 線形サーチであるため性能は線形に劣化

- **Open vSwitch**
  - フローキャッシュ（ハッシュ）
    - MegaFlow

  → 本研究で検討

- **Packet Classification**
  - トライ木
    - Hierarchical Trie, Set-Pruning Trie
  - 決定木（n次元点位置決定問題への還元）
    - HyperSplit [Qi et al. 2009]
    - Efficut [Vamanan et al. 2010]

  → 本研究で検討

  → 決定木構築まで時間がかかる

アルゴリズム検討：
ハッシュによるアプローチ

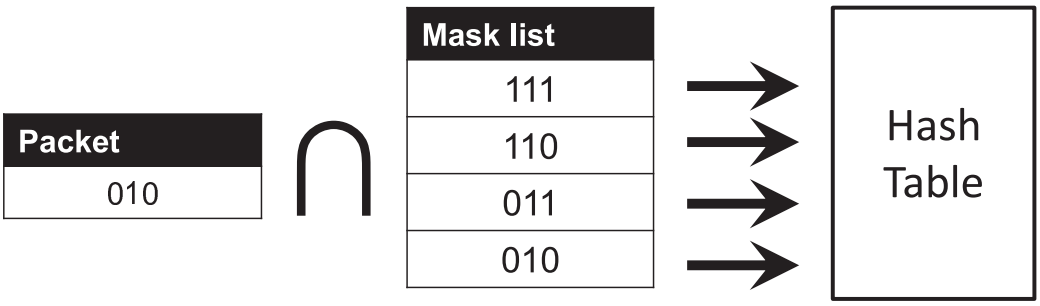
Mask List (M-List)

最悪メモリアクセス回数量： O(M)

- Open vSwitchのフローキャッシュ検索で実装される
- Entryの全種類のMaskに対してハッシュを計算
- (Entry数) >> (Maskの数)である為，高速化が可能

※MはMaskの種類数

Entry	
R1	000
R2	001
R3	01*
R4	101
R5	10*
R6	*00
R7	*0*



アルゴリズム検討：
トライ木によるアプローチ

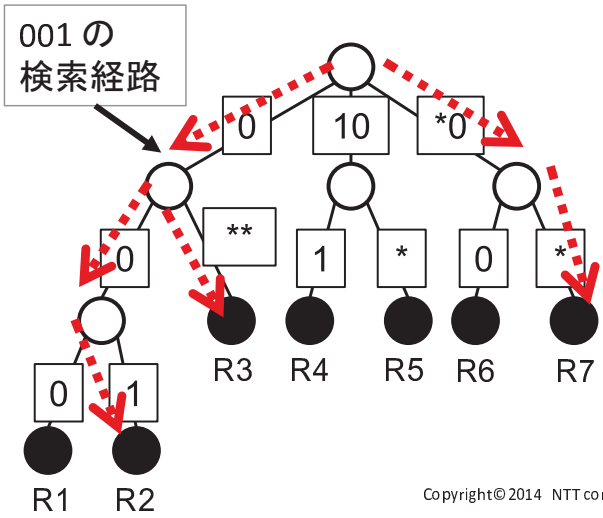
Hierarchical Trie （H-Trie）

最悪メモリアクセス回数量： O(2N)

最悪記憶容量： O(N)

- 0, 1, *の3種類で分岐するトライ木
- バックトラッキングによりマッチする全ノードを辿る.
- 縮約すると (ノード数) < 2N となり，検索はO(2N)

Entry	
R1	000
R2	001
R3	0**
R4	101
R5	10*
R6	*00
R7	*0*



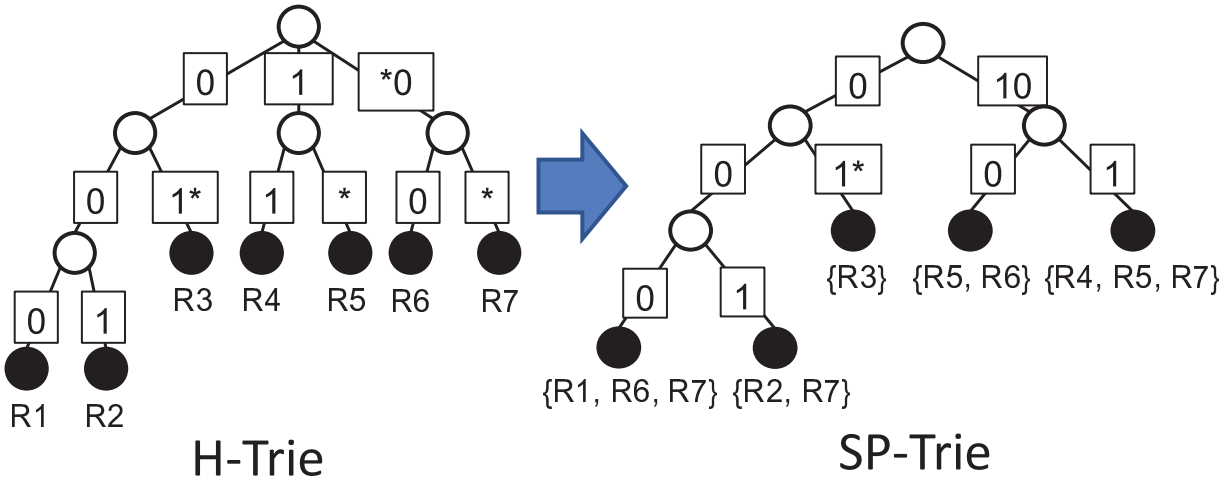
Set-Pruning Trie （SP-Trie）

最悪メモリアクセス回数量： O(L)

最悪記憶容量： $O(2^L)$

- H-Triesのバックトラックを削除
- 辿る毎にマッチする可能性のあるEntry集合を縮小
- 検索はO(L)だが，最悪空間計算量は$O(2^L)$

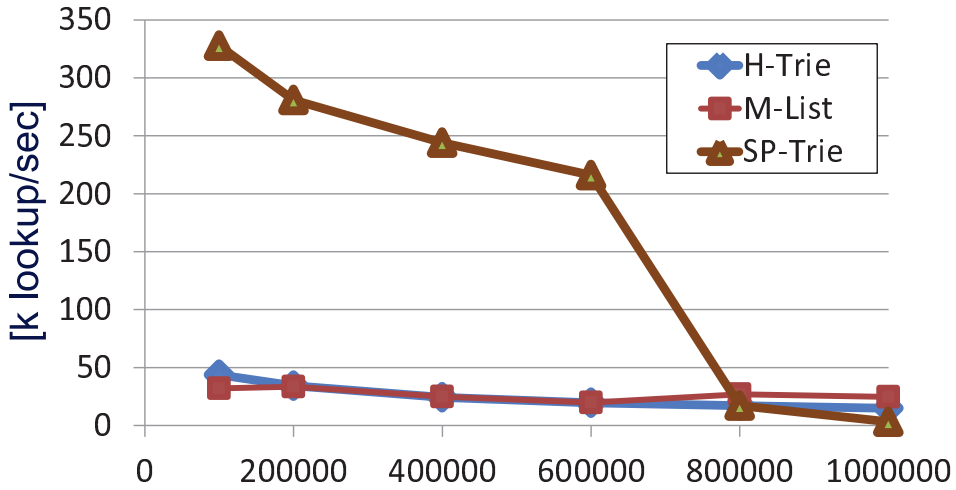
Entry	
R1	000
R2	001
R3	01*
R4	101
R5	10*
R6	*00
R7	*0*



NTT

H-Trie vs. SP-Trie vs. M-List

- 3つの手法を比較
  - SP-Trieが高速だが，エントリ増加で急激に性能劣化
  - M-List, H-Trieは性能劣化が少ないが，低速
  - SP-TrieとH-Trieを組み合わせた手法を検討



Entry数に対するLookup性能

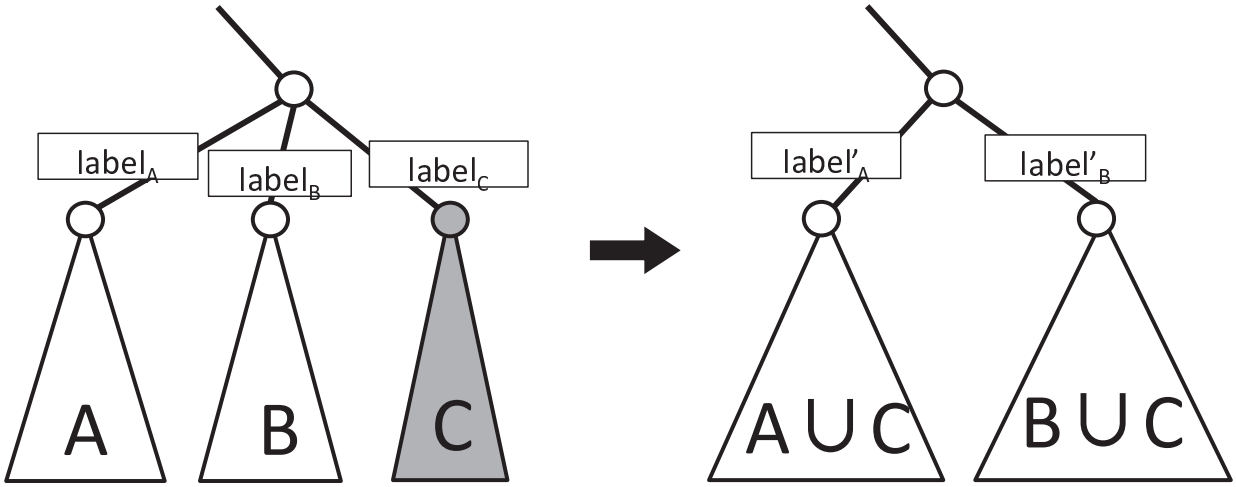
条件
- key長: 32bit
- Mask数：100
- key: rand()
- Mask: (rand()|rand())

環境
- OS: Ubuntu 12.04LTS
- GCC version 4.6.3
- CPU: Intel Xeon E5-2660
- DDR3-1600 ECC 64GByte
- NIC: Intel X520-DA2

NTT

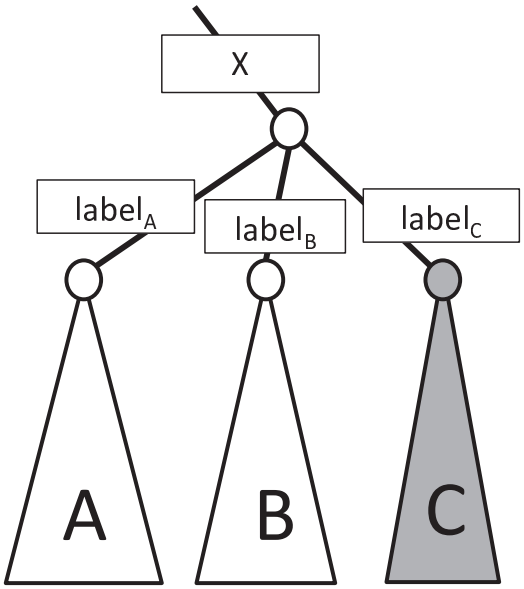
組合せ（H-TrieとSP-Trie）

- H-Trieのバックトラックはある頂点から（AまたはB）とCの部分木の2つを辿ることが原因
- Cの部分木をA，Bにマージする
- 全頂点でこの操作を行うと　H-Trie　⇨　SP-Trie



組合せの実現手法

- 部分木Cに含まれるエントリを分割し，A, Bに登録.
- 追加・削除を順に行えば，検索と並列実行可能



C_i	X	label$_C$ = "*xxxx"	C_i

C_i^0	X	label$_C^0$ = "0xxxx"	C_i
C_i^1	X	label$_C^1$ = "1xxxx"	C_i

組合せの利点・欠点

- 利点
  - メモリの上限に合わせ，高速化可能
  - 一旦，H-Trieのようにエントリを追加する為，Addが高速
- 欠点
  - どのエントリを分割するのが効率的か，決定が難しい．⇨　今回は根から幅優先順で実施

トライ木の実装手法検討

実装による高速化

- **実装による高速化 ⇨ キャッシュヒット率の向上**

- 子ノードを実メモリ上の近くに配置
  - Double Array [Aoe et al. 1989]
  - van Emde Boas layout [Brodal et al. 2002]

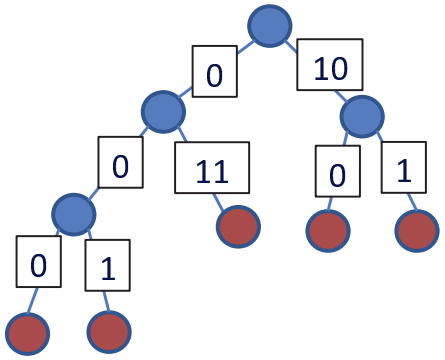
検討①

- 省メモリ化
  - 簡潔データ構造(LOUDS) [Jacobson et al. 1989]

検討②

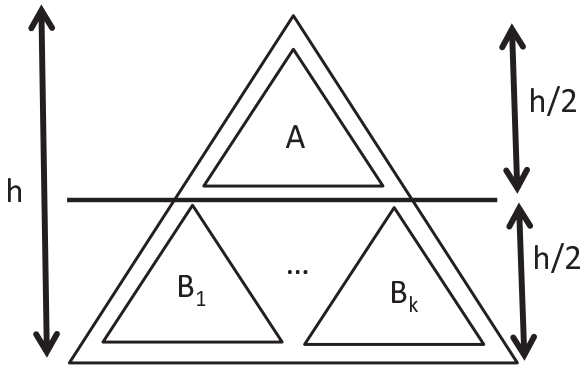
検討①: Double Array と vEB layout

- Double Array（DA）により配列上にトライ木を実装
- van Emde Boas layout（vEB）によりキャッシュの最適化



label	0	10	0	11	0	1	0	1
base	2	4	6	-1	-2	-3	-4	-5

Double Array



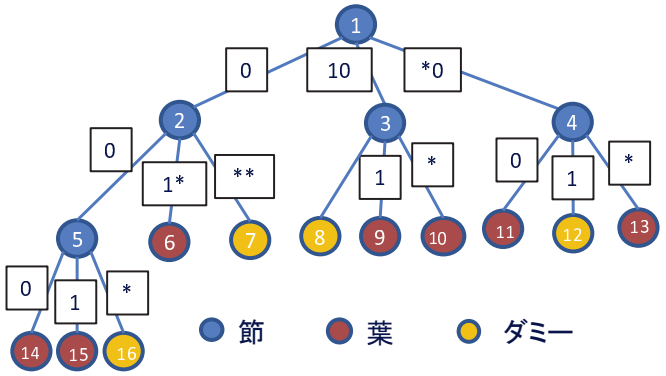
A	B_1	...	B_k

van Emde Boas layout

検討②：簡潔データ構造（LOUDS）

- 簡潔データ構造の一つであるLOUDSを拡張
  - ポインタ分省メモリ化（約8byte/node削減）
  - 拡張により，検索はRank操作と簡単な演算のみ

Entry
000
001
01*
101
10*
*00
0



child_id = $rank_{a1}(id) * 3 + n$

ノード番号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
bit列: a	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

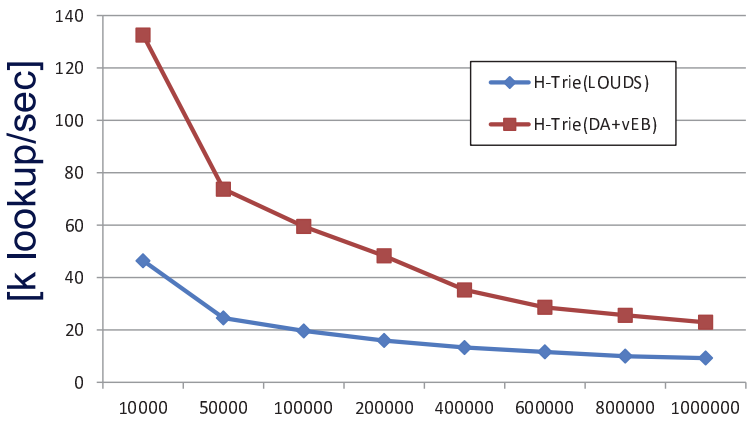
data_index = $rank_{b1}(rank_{a0}(\mathrm{id}))$

ノード番号	6	7	8	9	10	11	12	13	14	15	16
bit列: b	1	0	0	1	1	1	0	1	1	1	0

NTT

DA+vEB vs. LOUDS

- H-Trieを2つの手法で実装し比較
  - DA+vEBが高速
  - LOUDSは検索速度がDA+vEBの1/3だが，約25%省メモリ
- 検索速度がより重要であるため，ソフトウェアOpenFlowスイッチではDA+vEBの実装が有効



Entry数に対するLookup性能

条件
- key長: 32bit
- Mask数：100
- key: rand()
- Mask: (rand()|rand())

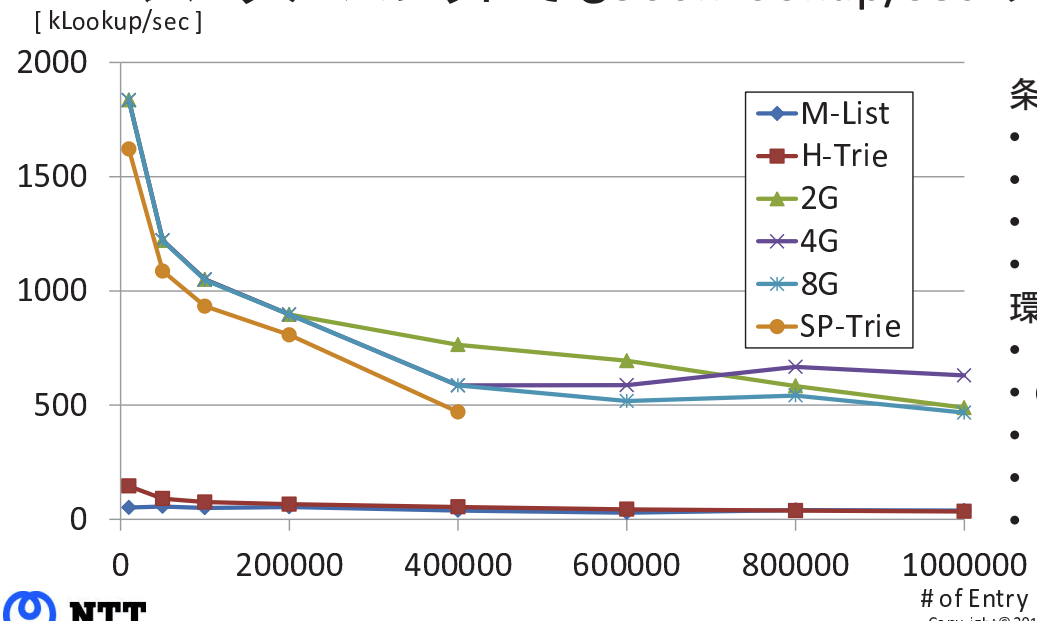
環境
- OS: Ubuntu 12.04LTS
- GCC version 4.6.3
- CPU: Intel Xeon E5-2660
- DDR3-1600 ECC 64GByte
- NIC: Intel X520-DA2

NTT

統合・評価

統合・評価

- H-TrieとSP-Trieの組み合わせ手法をDA+vEBで実装
- ランダムな値で性能測定
- シングルスレッドでも500kLookup/secの性能.



条件
- key長: 32bit
- Mask数 : 100
- key: rand()
- Mask: (rand()|rand())

環境
- OS: Ubuntu 12.04LTS
- GCC version 4.6.3
- CPU: Intel Xeon E5-2660
- DDR3-1600 ECC 64GByte
- NIC: Intel X520-DA2

まとめ

- Bitmask対応の高速なフロー検索手法をハッシュとトライ木を用いた2つアプローチで検討を行い，H-TrieとSP-Trieを組み合せる手法が有用であることを示した.
- トライ木による検索木をDA+vEBとLOUDSで実装した. DA+vEBが高速であり，フロー検索の実装として有効であった.
- 上記2つの手法を組合せ，ランダムな入力に対しシングルスレッドで500kLooup/secの性能を得た.
- 今後，さらに並列化等の高速化を行い，実際のソフトウェアOpenFlowへの組み込む.

NTT

NTT