# Two Dimensional Router: Design and Implementation

Shu Yang* Mingwei Xu* Dan Wang† Dan Li* Jianping Wu*
*Tsinghua University †The Hong Kong Polytechnic University

## Abstract

Packet classification has attracted research attentions along with the increasing demands for more flexible services in the Internet today. Among different design approaches, hardware based classification is attractive as it can achieve very fast performance. We present our design and implementation of a two dimensional router (TwoD router). It makes forwarding decisions, in hardware level, based on both destination and source addresses. This can fundamentally increase routing semantics to support services beyond destination-based routing. Our TwoD router is also motivated by our effort towards developing two dimensional IP routing.

With one more dimension, the forwarding table grows explosively. Theoretically, the storage footprint increases from $O(N)$ (the number of destination prefixes) to $O(N^2)$. It is no longer possible to fit such forwarding table into TCAM, the de facto router standard. In this paper, we propose a new forwarding table structure through a neat separation of TCAM and SRAM where we harvest the speed of TCAM and the storage/flexibility of SRAM. Under this structure, we first develop schemes to maintain forwarding correctness. We then develop algorithms to further compress TCAM and SRAM storage. With a pipelining scheme, we show that the lookup speed of our TwoD router is the same to the conventional routers. We also develop algorithms that minimize memory rewrites during update operations. We evaluate our design with an implementation on a commercial router, Bit-Engine 12004, using real data sets to support two practical services, policy routing and load balancing of CERNET2. Our design does not need new hardware. The evaluation results show that our TwoD router can be practical for ISPs like CERNET2.

## 1 Introduction

To provide reachability service, conventional Internet routers classify packets based on destination address. Such one dimensional router is adequate for destination-based packet delivery. Nevertheless, there are increasing demands for such services as policy routing, security, traffic engineering, quality of service, etc. The paucity of routing semantics makes innovation in developing higher level services more difficult or restricted to limited scope, e.g., on edge routers only.

There are many studies on packet classification, among which 5-tuple (destination and source addresses, destination and source ports, transport layer protocol) [17] and 2-tuple (destination and source addresses) [5][38][22] have attracted most attention. Many studies are software-based solutions. It has limitation to scale to network wide deployment and support services with high performance requirements.

In this paper, we develop routers that have the 2-tuple packet classification in hardware level. More specifically, the forwarding decision is based not only on the destination address, but also on the source address. We choose to develop such router because 1) 2-tuple packet classification is easier to start; 2) destination address provides reachability information, and source address provides identity information. We believe that destination and source addresses have a higher priority in routing semantics. We show that our TwoD router achieves the same lookup speed as conventional routers. We further show examples, where TwoD router can be used to easily support services as policy routing and load balancing.

This router is also motivated by our on-going work on deploying Two Dimensional-IP (TwoD-IP) routing for China Education and Research Network 2 (CERNET2) [45], which is the largest IPv6 network worldwide. With TwoD-IP routing, we can easily deploy policy routing, security services, traffic engineering, which have been looking forwarded by CERNET2 for a long time [46].

There are many challenges to make this router real. For hardware-based forwarding, the de facto standard of the routers is TCAM-based forwarding. It achieves fast and constant lookup time. TCAM, however, has low capacity, large power consumption and high cost [26]. The

largest TCAM chip in market can only accommodate 1 million IPv4 prefixes [27].

When designing a TwoD router, the immediate change it brings about is the forwarding table size. More specifically, the Forwarding Information Base (FIB) stored in TCAM will tremendously increase. This is because a straightforward implementation, i.e., using the traditional Cisco Access Control List (ACL) structure (we call it *ACL-like structure* thereafter), means that the FIB table changes from $\{\text{destination}\} \rightarrow \{\text{action}\}$ to $\{(\text{destination, source})\} \rightarrow \{\text{action}\}$. Let $N$ be the number of destination prefixes and $M$ be the number of source prefixes. In the worst case, the number of TCAM entries can be $O(N \times M)$. This structure increases TCAM size by an order and a practical consequence is that TCAM cannot hold entries of such scale. Note that the number of destination prefixes in current backbone routers is 400,000 [1]. If a TwoD router is implemented by an ACL-like structure, it is beyond the TCAM storage even with a few tens of source prefixes.

In this paper, we design a new forwarding table structure called FIST (**FI**B **S**tructure for **T**woD-IP). The key of FIST is a neat separation of TCAM and SRAM. SRAM has larger memory and it is 10 times cheaper and consumes 100 times less energy than TCAM. In FIST, TCAM contributes fast lookup and SRAM contributes a large memory space for nexthop storage. This separation reduces the TCAM storage from $O(N \times M)$ to $O(N+M)$.

We solve many challenges under this new FIST structure. First, a straight-forward implementation of this separation will leave empty cells in SRAM table. We develop a scheme to saturate the table and prove the correctness of FIST in handling forwarding operations. Second, we develop algorithms for further TCAM and SRAM storage compression. Especially, SRAM is flexible and we develop deduplication schemes that substantially reduce SRAM footprint. Third, we develop efficient pipeline schemes for nexthop lookup operation that achieves the same performance to conventional routers. We further propose a color tree structure to organize the entries and develop algorithms to minimize the number of rewrites on SRAM in an update operation.

Besides the architecture choice and algorithmic designs, we tune the TwoD router with important implementation designs. More specifically, we present a non-homogeneous FIST structure, a special treatment for default source prefix, and some important parameter choices. These further improve TwoD router in practice.

We implement the FIST on a commercial router, BitEngine 12004. We only need to redesign the hardware logic, and we do not need new hardware. We carry out comprehensive evaluations using the real topology, FIB prefix and traffic data from CERNET2, in two practical scenarios, policy routing and load balancing. The results show that our TwoD router can achieve linecard speeds, with very small TCAM and SRAM footprint.

The paper is organized as follows: We present background and related work in Section 2. Section 3 is devoted to TwoD router storage design. We introduce the FIST structure and we prove its correctness for the forwarding operation. We further present our TCAM and SRAM compression schemes. Section 4 is devoted to TwoD router operation designs: the nexthop lookup and update operations. We present our implementation designs in Section 5, which improve TwoD router in practical situations. Section 6 show our implementation on a commercial router. We evaluate the TwoD router in Section 7. In Section 8 we discuss some current limitations. Finally, we conclude our paper in Section 9.

## 2 Background and Related Work

We first present some background on hardware packet processing in a router. We show a common implementation in Fig. 1. After a packet arrives on a router, it will traverse the datapath on a linecard, which include ACL (Access Control List), PBR (Policy Based Routing) and finally FIB. ACL and PBR are used for different levels of packet classification, and mostly implemented at powerful edge routers. In some implementations, ACL and PBR are merged into one module [11] and some simple routers only have FIB. The key that there are so many separated modules instead of one FIB is that we do need packet classification, yet the complexity of packet classification makes it difficult for FIB to support them directly. Therefore, the FIB is made up with only destination-based prefixes, and ACL and PBR contains necessary classification rules. If a packet matches the rules in ACL or PBR, they will be filtered (or handed in to control plane for further processing) and forwarded. Otherwise, it goes to the FIB for destination-based forwarding. The conventional FIB structure is shown in Fig. 2(a). The prefixes are stored in TCAM and the nexthop actions are stored in SRAM.

Clearly, if we may build more routing semantics into FIB itself, it makes the router logically more sound, and possibly less messier of the rules in the ACL/PBR modules. In this paper, we show a first step by including source address (which we think contains the richest information besides destination address) in the FIB.

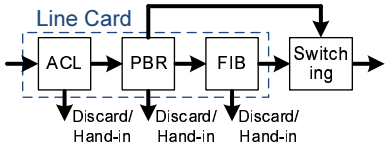
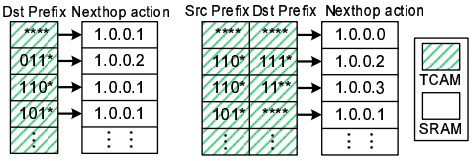



Figure 1: The datapath that packets traverse

Packet classification is an important topic throughout the Internet history [37][13][40][41]. For software-based solutions, the simplest solution is linear search. Others




(a) FIB (b) ACL-like

Figure 2: Conventional FIB and ACL-like structures

can be classified into decision tree, tuple search and decomposition based solutions. Decision tree techniques [11][34] use tree structure to prune the search space and find a single rule. In [42], a novel structure for cutting the address space is proposed to efficiently construct a tree. Tuple search techniques [36][35][5] exploit that the number of unique prefix length is much smaller than prefix number. These techniques thus set up hashing structure in each space partitioned based on the prefix length. Decomposition techniques [19][12][43] break the classification process into several parallel searches on single dimensions. In [19], each dimension is matched separately and returns a bit vector, that will be ANDed together. In [43], each single dimension is searched in SRAM using a novel encoding mechanism, and concatenation of the encoded results in single dimensions is searched in TCAM. In [39], optimized data structures such as Bloom Filter Arrays are used to improve the lookup efficiency. Software based solutions suffer from performance and are less popular than hardware-based solutions.

For hardware (TCAM) based solutions, they can achieve constant lookup time, yet they are limited by the TCAM capacity [27][26]. Various compression or aggregation schemes have been proposed [21][38][25]. Most studies are based on the ACL-like structure. For example, an encoding algorithm is proposed to reduce the width of TCAM [20]. In [25], a non-prefix approach to compress TCAM is studied. Intrinsically, conventional FIB structure suffers from multiplicative effect [43], and this leads to storage explosion [40][39]. For example, a straightforward TwoD router design using ACL-like structure is in Fig. 2(b). We observe that conventional FIB structure leads to a 'fat' TCAM and a 'thin' SRAM.

There are studies proposing more compact structures to reduce the multiplicative effect [26]. For example, in [27], it first lookups in a one dimensional table storing destination prefixes, and outputs a sub-table storing source prefixes, where the actions can be found. Thus, it can merge different sub-tables if they are the same. In [28], SPliT is proposed which exploits SRAM. The width of SRAM is extended such that each prefix points to multiple actions. Thus sub-tables can be merged without having their actions the same. It still does not intrinsically solve the multiplicative effect. There is a recent work that focuses on reducing the power consumption [23]. The TCAM size is not reduced. Our work specifically considers source address, which we believe can be put in a higher priority in packet classification and we fully eliminate the multiplicative effect in TCAM.

Our work is also inline with our on-going effort in developing Two Dimensional IP Routing for CERNET2 [46]. We have already deployed source functions on some edge routers [44] to achieve better security. We look forward for a network wide deployment of two dimensional routers. It makes design and development of a wider range of services much easier with such support.

| # | Destination prefix | Source prefix | Nexthop action | # | Destination prefix | Source prefix | Nexthop action |
|---|---|---|---|---|---|---|---|
| 1 | **** | **** | 1.0.0.1 | 11 | 110* | 01** | 1.0.0.2 |
| 2 | **** | 101* | 1.0.0.0 | 12 | 101* | **** | 1.0.0.1 |
| 3 | **** | 11** | 1.0.0.2 | 13 | 101* | 101* | 1.0.0.0 |
| 4 | **** | 01** | 1.0.0.0 | 14 | 101* | 11** | 1.0.0.2 |
| 5 | 011* | **** | 1.0.0.2 | 15 | 101* | 01** | 1.0.0.0 |
| 6 | 110* | **** | 1.0.0.1 | 16 | 11** | **** | 1.0.0.2 |
| 7 | 110* | 111* | 1.0.0.2 | 17 | 11** | 11** | 1.0.0.3 |
| 8 | 110* | 101* | 1.0.0.0 | 18 | 10** | **** | 1.0.0.2 |
| 9 | 110* | 100* | 1.0.0.2 | 19 | 10** | 100* | 1.0.0.2 |
| 10 | 110* | 11** | 1.0.0.3 | 20 | 10** | 11** | 1.0.0.3 |

Table 1: A Two Dimensional Forwarding Example

# 3 FIST: the TwoD Routing Table

## 3.1 The TwoD Forwarding Rule

In conventional router, longest matched first (LMF) rule is used to decide which destination prefix will be matched. Here, we first present the definition of the forwarding rules that is used in two dimensional routing. Let $d$ and $s$ denote the destination and source addresses, $p_d$ and $p_s$ denote the destination and source prefixes for $d$ and $s$. Let $a$ denote an action, more specifically, the nexthop corresponding to $p_d$ and $p_s$. The storage structure of a TwoD router is a table of entries of 3-tuple $(p_d, p_s, a)$.

**Definition 1.** ***TwoD forwarding rule****: Assume a packet with s and d arrives at a router. The destination address d should first match $p_d$ according to the LMF rule. The source address s should then match $p_s$ according to the LMF rule among all the 3-tuples given that $p_d$ is matched. The packet is then forwarded to next hop a.*

Our rule is defined based on the following principles: 1) conflict avoidance: it has been shown [22] that if matching the source and destination addresses at the same priority, LMF rule cannot decide. Even using the first-matching-in-table tie breaker still results in loops, and resolving the conflicts is NP-hard. 2) compatibility: matching destination prefixes first emphasizes on connectivity and is compatible with the current destination-based architecture. If no source prefix is involved, our rule naturally regresses to traditional forwarding rule.

## 3.2 FIST Basics and Correctness

### 3.2.1 FIST Basics

The key idea is a separation of TCAM and SRAM (see Fig. 3). In this separation, the destination and source

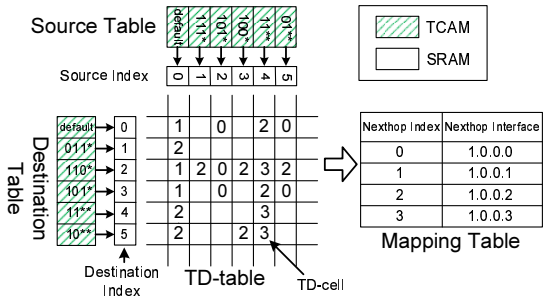



Figure 3: FIST: A forwarding table structure for TwoD-IP

prefixes are stored in TCAM, with an offset table pointing to the nexthop table stored in SRAM. As the nexthop information is long, we have another mapping table so that the main SRAM nexthop table only stores an index.

Formally, we design Forwarding Information Base Structure for Two Dimensional Router (FIST). FIST has two tables stored in TCAM and two tables stored in SRAM. In TCAM, one table stores the destination prefixes mapping to a *destination index* (we call the table *destination table* thereafter), and one table stores the source prefixes mapping to a *source index* (we call the table *source table* thereafter). One table in SRAM is a two dimensional table that stores the indexed nexthop of each rule (we call it *TD-table* thereafter) and we call each cell in the array *TD-cell*. The destination and source indexes in TCAM point to a TD-cell in SRAM. The other table in SRAM stores the mapping relations of index values and next hops (we call it *mapping-table* thereafter).

For each rule $(p_d, p_s, a)$, $p_d$ is stored in the destination table, and $p_s$ is stored in the source table. For the $(p_d, p_s)$ cell in the TD-table, there stores an index value. From this index value, $a$ is stored in the corresponding position of the mapping table. Note that the index value is much shorter than the nexthop information $a$.

We show an example in Fig. 3. For (110*,11**, 1.0.0.3), 110* is stored in the destination table and points to destination index 2, that is associated with the $2_{nd}$ row; and 11** is stored in the source table and points to source index 4, that is associated with the $4_{th}$ column. In the TD-table, the cell $(110^*, 11^{**})$ that corresponds to $2_{nd}$ row and $4_{th}$ column has index value 3. In the mapping table, the next hop with index value 3 is 1.0.0.3.

With FIST, The FIST TCAM storage is $O(N+M)$ bits. The FIST SRAM storage is $O(N \times M)$ bits.

Clearly, FIST migrates the "multiplication" factor into SRAM, rather than eliminate it. Such migration is based on the following facts: 1) TCAM storage capacity is much smaller than SRAM; 2) TCAM is 10-100 times more expensive than SRAM; 3) TCAM consumes 100+ times more power than SRAM [21][9][4].

This migration does not slow down the forwarding speed. The dominant factor for forwarding speed is TCAM lookup, which FIST maintains. In FIST, there are additional SRAM accesses. We discuss this shortly and develop a pipelining scheme so that the router performance is the same with that of conventional routers.

SRAM is more flexible than TCAM. We will show that we can develop compression algorithms in SRAM to further reduce TD-table storage. Before all these, we first discuss FIST correctness.

### 3.2.2 FIST Correctness: FIST Establishment and TD-cell Saturation

For the sake of conciseness, we only focus on TD-table establishment. We establish a TD-table by inserting the entries (e.g., Table 1) into an empty TD-table. Fig. 3 shows the TD-table after inserting the entries in Table. 1.

Note that after insertion, there will be empty cells. Consider a packet with destination address 1011 and source address 1111 arrives at the router. According to TwoD forwarding rule, the destination prefix 101* will first be matched. There are four rules (including the default rule) associated with the destination prefix 101*. Source prefix 11** will then be matched. This leads to rule $(101^*, 11^{**}, 1.0.0.2)$. With the new structure, however, destination prefix 101* will be matched and source prefix 111* will be matched. Unfortunately, cell $(101^*, 111^*)$ ($3_{rd}$ row and $1_{st}$ column) in TD-Table does not have any index value. Intrinsically, for prefix pairs $(p_d, p_s)$, if there exists a source prefix $p'_s$ that is longer than $p_s$, cell $(p_d, p'_s)$ rather than $(p_d, p_s)$ will be matched.

**Definition 2.** ***Conflicted cell****: For a TD-cell $(p_d, p'_s)$, if there is rule $(p_d, p_s, a)$ where $p_s$ is a prefix $p'_s$, and we cannot find an action $b$ such that $(p_d, p'_s, b)$ itself is a rule, we call $(p_d, p'_s)$ a conflicted cell.*

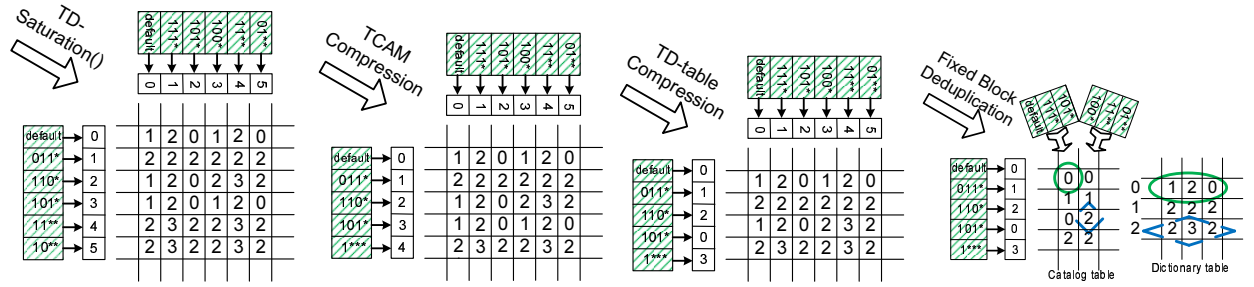
To address the problem, we develop algorithm TD-Saturation() to saturate the conflicted cells with appropriate index value. As an example, using this algorithm, the TD-table of Fig. 3 becomes Fig. 4(a).

**Algorithm 1:** TD-Saturation($\mathcal{R}$)

```
1  begin
       // R is the set of TwoD forwarding rules
2      foreach p_d, p_s do
3          if ∄(p_d, p_s, a) ∈ R then
4              S = {(p̃_s, p̃_d, ā) ∈ R | p̃_d = p_d}
5              S' = {(p̂_s, p̂_d, â) ∈ S | p̂_s is a prefix of p_s}
6              Find (p̂_s, p̂_d, â) ∈ S', ∀(p'_d, p'_s, a') ∈ S', p'_s is a
               prefix of p̂_s
7              Fill the cell (p_d, p_s) with index value of â.
```

**Theorem 1.** *FIST (with* TD-Saturation*()) correctly handles the rule defined in Definition 1.*

*Proof.* If a packet matches a conflicted cell $(p_d, p_s)$. $\mathcal{S}$ contains all rules given $p_d$ is matched. $\tilde{p_s}$ is a prefix of $p_s$, thus the packet also matches rules in $\mathcal{S}'$. According to Line 6, $(\hat{p_s}, \hat{p_d}, \hat{a})$ is the longest match in source given $p_d$ is matched. So $(p_d, p_s)$ should be set with the index of $\hat{a}$ according to Definition 1. □




(a) Apply TD-Saturation() (b) TCAM compression (c) SRAM - optimal TD-table (d) SRAM - fixed block deduplication

Figure 4: FIST forwarding table storage

## 3.3 Forwarding Table Compression

The FIST structure provides us a framework that maintains the processing speed in TCAM and migrates the storage to SRAM. Under this framework, we further minimize both the TCAM tables and SRAM tables. We first formally define equivalence class of TwoD tables. As such, we can select a TwoD table in the equivalence class that has the minimum size.

Let $P_d$, $P_s$ be the set of destination, source prefixes respectively. Let $f_r(p_d), p_d \in P_d$ (or $f_c(p_s), p_s \in P_s$) be a mapping function that maps a destination (or source) prefix $p_d$ (or $p_s$) to a destination index (or source index). Let $TD(x,y)$ denote the cell in $x_{th}$ row and $y_{th}$ column of TD-table. We use a 5-tuple $\{P_d, P_s, f_r(\cdot), f_c(\cdot), TD(\cdot,\cdot)\}$ to denote a TwoD forwarding table.

**Definition 3.** $\{P'_d, P'_s, f'_r(\cdot), f'_c(\cdot), TD'(\cdot,\cdot)\}$ *is* ***equivalent*** *to* $\{P_d, P_s, f_r(\cdot), f_c(\cdot), TD(\cdot,\cdot)\}$*, if for any destination address d and source address s, d matches* $p_d$ *in* $P_d$ *and* $p'_d$ *in* $P'_d$*, s matches* $p_s$ *in* $P_s$ *and* $p'_s$ *in* $P'_s$ *according to LMF rule, then* $TD(f_r(p_d), f_c(p_s)) = TD'(f'_r(p'_d), f'_c(p'_s))$.

For a given forwarding table, our objective is to find an equivalent forwarding table with the minimum storage space. We discuss TCAM and SRAM separately.

### 3.3.1 TCAM Compression

For TCAM compression, there exists a dynamic programming based algorithm Optimal Routing Table Constructor (ORTC) [10], which computes the minimized TCAM for a set of prefixes in one dimensional router.

In our scenario, we apply ORTC separately to destination and source tables in TCAM. There are two problems. First, in a one dimensional routing table, a prefix leads to a scaler (the nexthop), yet in a TwoD routing table, a prefix leads to a vector. Second, we need to study the quality of application of ORTC, i.e., we show that it also leads to minimum storage in TCAM.

To map this vector to a scaler, we preprocess the vector with a collision-resistant hash function. We use SHA-1[1]. ORTC is then applied.

[1]Here, with SHA-1, the collision probability is many orders of magnitude smaller than hardware error rate [32].

**Theorem 2.** *Applying ORTC twice on FIST, one on destination table and the other on source table, leads to the minimum combined TCAM storage.*

*Proof.* We give the algorithm and proof in [47]. □

For example, after TCAM compression, the forwarding table in Fig. 4(a) becomes Fig. 4(b).

### 3.3.2 SRAM Compression

There are two tables in SRAM, where the TD-table dominates. We thus focus on minimizing the TD-table.

**Problem 1.** ***Optimal TD-table Compression:*** *Given* $\{P_d, P_s, f_r(\cdot), f_c(\cdot), TD(\cdot,\cdot)\}$*, find an equivalent forwarding table* $\{P'_d, P'_s, f'_r(\cdot), f'_c(\cdot), TD'(\cdot,\cdot)\}$ *such that the storage space in the TD-table, i.e.,* $|\{f'_r(p'_d)|p'_d \in P'_d\}| \times |\{f'_c(p'_s)|p'_s \in P'_s\}|$ *is minimized.*

To find the optimal TD-table compression, we simply compress the TD-table by merging duplicated rows (columns). In a one dimensional router, merging two entries requires two destination prefixes to be aggregatable. It is not necessary in FIST structure since we can merge the rows (columns) as long as we make their destination (source) indexes the same.

We call two rows (columns) in TD-table *duplicated* rows (columns ) if $\overrightarrow{TD(f_r(p_d),\cdot)} = \overrightarrow{TD(f_r(p'_d),\cdot)}$ ($\overrightarrow{TD(\cdot, f_c(p_s))} = \overrightarrow{TD(\cdot, f_c(p'_s))}$).

**Theorem 3.** *Eliminating the duplicated rows and columns computes the optimal TD-table compression.*

*Proof.* We give the proof in [47]. □

For example, after eliminating duplicated rows (columns), the table in Fig. 4(b) becomes Fig. 4(c).

Although we can find the optimal TD-table compression, rows that are entirely duplicated are rare, as such, this alone has a very small compression ratio. Note that SRAM is much more flexible than TCAM. We can take this advantage to develop improvement schemes. We observe there are abundant duplications if we only extract a sub-chunk of a row. Intuitively, we can compress them. Our goal is therefore to search the TD-table to find redundant patterns, extract them, and use single pointers to replace them. Similar methods in data compression for disk and file systems can be found in [24].

**Problem 2.** ***Optimal SRAM Compression:*** *Given a TD-table, find a minimized representation of it such that 1) it only contains nexthop index and pointers to other positions in itself; 2) we can find a TD-cell in constant time.*

This problem is NP-complete (see proof in [47]).

Due to the complexity of the problem, we solve it with a heuristic - *fixed block deduplication*. The basic idea is to cut a full row into sub rows and merge them. The probability of duplicated sub rows becomes high.

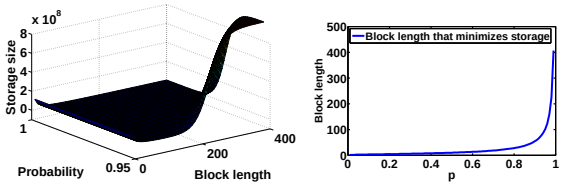
Merging sub rows requires some modification on TD-table. Fortunately, SRAM is much more flexible than TCAM to incorporate changes.

A *sub-row* is a continuous group of cells in a row. Let $\tilde{w}$ ($0 < \tilde{w} < M$) be the length (number of cells) of a sub-row. This $\tilde{w}$ is predetermined (we will analyze $\tilde{w}$ later) for a TD-table. We separate the TD-table into a *catalog table* and a *dictionary table* (see an example in Fig. 4(d) where $\tilde{w} = 3$). The TD-table is divided into sub-row chunks. The dictionary table contains all unique chunks. The catalog table maps every chunk of sub-rows of the TD-table into a single cell, where the value in the cell is the index to the dictionary table. Note that TD-table and catalog table-dictionary table is a one-one mapping.

We develop a deduplication procedure to construct the new catalog-dictionary tables from a TD-table. A formal flow chart is in [47]. Basically, we scan the TD-table, and extract all sub-rows. For each sub-row, we first compute its *fingerprint*, using SHA-1 function. With bloom filter [8], we can judge whether the sub-row is a duplicated one or not. If bloom filter judges it is, we search in a data structure called *fingerprint store*, which organizes all detected fingerprints with $< fingerprint, r, k >$ triples, where $r$ is the sub-row index in the dictionary table, and $k$ is the number of sub-rows that are hashed to $fingerprint$. If we can not find it in fingerprint store (due to the false positive probability of bloom filter) or bloom filter judges it is not duplicated, then we insert the sub-row into the dictionary table, and a new triple into the fingerprint store. Using the search result, we fill the corresponding cell of catalog table with the sub-row index. We explain how bloom filter and fingerprint store can be implemented in practice in Section 5.3.

**Theoretical Analysis:** Let $p_r$ ($p_c$) be the probability that two cells in the same row (or column) are identical. Let $p_u$ be the probability that two cells in different rows and different columns are identical. We assume that $p = p_r = p_c \gg p_u$, and $N > M \gg 1$. We present formal analysis on the catalog-dictionary table structure in [47]. Two main analytical results can be summarized as: 1) we should cut rows rather than columns and 2) for a sparser TD-table, the block length should be larger; For a denser TD-table, the block length should be smaller.

As an example, in Fig. 5(a), we show the storage size as a function of block length and $p$, with $N = 100,000$, $M = 10,000$ and the size of each cell in catalog and dictionary tables to be one unit. We can see that at a fixed $p$, storage size first decreases with block length, because of deflation of catalog table; and then increases, because of inflation of dictionary table. In Fig. 5(b), we show the block length that minimizes the storage size. We can see that if $p$ is large, i.e., TD-table is sparse, the block length should be large. This is because for sparse TD-table, larger block length reduces the catalog table while increasing dictionary table a little.




(a) Storage size (b) Optimal block length

Figure 5: Relation between storage size and block length, $p$

## 4 TwoD Routing Table Operations

We investigate two operations of the TwoD routing table, forwarding lookup and routing table update.

### 4.1 FIST Lookup

We first present the basic lookup steps and then show a pipeline lookup. We will show that the pipeline lookup achieves the same performance as the conventional routers for each lookup operation.

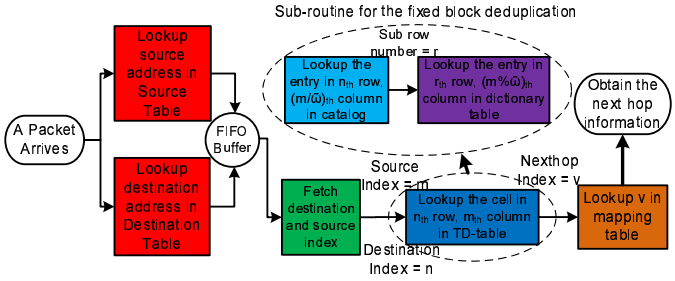
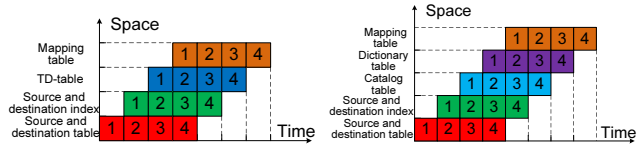



Figure 6: Lookup action in FIST

The lookup action is shown in Fig. 6. When a packet arrives, the router matches the destination and source prefixes in parallel in the destination and source tables in TCAM. This parallelism is possible since we have a saturated TD-table. The destination table and source table then each outputs the SRAM addresses that point to the destination index and source index. The SRAM addresses are passed to an FIFO buffer, which resolves the un-matching clock-rates between TCAM and SRAM. The router then obtains the destination index and source index. The router can thus identify the cell in the TD-table, and return the index value. Using this index, the




(a) Without fixed block deduplication (b) Within fixed block deduplication

Figure 7: Space-time diagram of the lookup process pipeline

router looks up the mapping table, and returns the nexthop information.

After applying the fixed block deduplication, the TD-table is separated into a catalog table and a dictionary table. Thus, we replace the TD-table lookup step with a new sub-routine. After obtaining the destination index and source index, the router can find an entry in the catalog table, and return a sub row index in the dictionary table. Using the sub-row and original source index, the router returns the nexthop index to the mapping table.

**Theorem 4.** *The lookup speed of FIST is one TCAM plus three SRAM clock cycles. Within fixed block deduplication, the speed increases by one SRAM clock cycle.*

*Proof.* The theorem is true because source and destination tables (indexes) can be accessed in parallel. □

As a comparison, the conventional destination-based routing stores destination prefixes in one TCAM, and accesses both TCAM and SRAM once during a lookup.

We develop a pipeline lookup process (see Fig. 7) for further amortizing each individual lookup operation. Pipelining itself is not new and almost all routers implement it today. Using the pipeline, the lookup speed of FIST can achieve one packet per clock rate.

**Observation 1.** *The lookup speed of the FIST routers (with pipelining) is the same as conventional routers.*

In practice, SRAM clock cycle is smaller than TCAM cycle [18].[2] Also, the bottleneck of a router is normally during delivering packets through the FIFO. These facts further validate that FIST lookup speed is comparable to conventional routers.

## 4.2 FIST Update

An update in routing table includes insertion, deletion and update. A big difference in TwoD router from one dimensional router, is that an update will no longer be a rewrite on a single value, but on multiple values that can even be a full row or column.

We consider three update actions: 1) insertion: Insert($p_d$, $f_r(p_d)$), Insert($p_s$, $f_c(p_s)$) and Insert($p_d, p_s, a$), 2) deletion: Delete($p_d$, $f_r(p_d)$), Delete($p_s$, $f_c(p_s)$) and Delete($p_d, p_s, a$), 3) update: Update($p_d, p_s, a$). To ease composition, we use Action($p_d$, $f_r(p_d)$), Action($p_s$, $f_c(p_s)$) and Action($p_d, p_s, a$) to denote them.

Action($p_d$, $f_r(p_d)$) happens when the reachability information of $p_d$ changes, e.g., BGP messages announce or withdraw $p_d$. Action($p_s$, $f_c(p_s)$) happens when router updates the policy of a source prefix. Action($p_d$, $f_r(p_d)$) updates a full row and Action($p_s$, $f_c(p_s)$) updates a full column in TD-table. The complexity cannot be reduced. We discuss solutions in practice in Section 5.

[2]TCAM is fast as it has extraordinary parallel process, yet also due to such parallelism, each individual access is slower than SRAM.

Action($p_d, p_s, a$) means a rule upon $p_d, p_s$ is changed. It can also incur changes in multiple values. For example, if we execute Update(101*, 11**, 1.0.0.3), both (101*, 11**) and (101*, 111*) have to change.

We develop an algorithm to find the minimum number of updates of TD-cells for Action($p_d, p_s, a$). Note that running this algorithm also needs time. From an architecture point of view, it is important to clarify the bottleneck: the rewrites on hardware or the update algorithm in CPU. The update algorithm are performed in the control plane. The access time of TD-cells (rewrite) is of the order of tens of nanoseconds (including the transition time on bus). If the running time of the algorithm does not match such speed, or there is a bottleneck on the bus between data and control planes, such algorithm should not be chosen. Nevertheless, from the design point of view, we present this algorithm. Our algorithm has the following properties: 1) it is optimal and 2) it can output the TD-cells to be rewritten one by one during the computation; this makes it possible that cell rewrites in the linecard and algorithm computation in the control plane to be pipelined. We first formally present the problem.

**Problem 3.** ***Optimal transformation:*** *Given a TD-table $TD(\cdot,\cdot)$ and* Action*($p_d, p_s, a$), find a new TD-table $TD'(\cdot,\cdot)$, such that $|\{(p_d, p_s)|TD(f_r(p_d), f_c(p_s)) \neq TD'(f_r(p_d), f_c(p_s))\}|$ is minimized.*

The key insight of our solution is that the cells associated with a destination prefix can be divided into two different groups. One is the conflicted cells and one is the rest. As such, we will first build color tree data structure to organize the cells. With this color tree, we will develop algorithms for insertion and deletion where only part of the nodes will be updated. Intrinsically, updates can be done only for a few nodes between confliction nodes. This color tree is also stored in the router control plane (more specifically in DRAM). Note that conventional router also stores data structure such as tries to organize prefixes. DRAM is much larger and cheaper, so the extra burden for our algorithms is acceptable. We will prove that our algorithms indeed minimize the computation cost and the number of cell rewrites.

**A Color Tree Structure and Update Algorithm**

We build a color tree $CT(p_d)$ for each destination prefix $p_d$. The tree includes all source prefixes in source table as nodes $Node(p_s)$. $Node(p_s)$ is an ancestor of $Node(p'_s)$ if $p_s$ is also a prefix of $p'_s$. The nodes are marked with two colors, black and white, where white nodes are those conflicted nodes in Definition 2 and the rests are black nodes. An example is shown in Fig. 8.

Let $\mathcal{B}(p_d) = \{Node(p_s)|\exists(p_d, p_s, a) \in \mathcal{R}\}$ be the set of black nodes, let $\mathcal{W}(p_d) = \{Node(p_s)|\neg\exists(p_d, p_s, a) \in \mathcal{R}\}$ be the set of white nodes. For example, in Fig. 8, we show a color tree $CT$(101*) for destination prefix 101* where $\mathcal{B}$(101*) = {$Node$(****),

$Node(01^{**})$, $Node(101^{*})$, $Node(11^{**})\}$ and $\mathcal{W}(101^{*}) = \{Node(100^{*}), Node(111^{*})\}$.

To compute optimal transformation of an update, we define *domain* of of a black node in color trees.

**Definition 4.** *In $CT(p_d)$, domain of $Node(p_s) \in \mathcal{B}(p_d)$ is $\mathcal{D}(p_d, p_s) = \{Node(p_s)\} \cup \mathcal{N}$, where $\mathcal{N} \subseteq \mathcal{W}(p_d)$ and $Node(p'_s) \in \mathcal{N}$ satisfies: 1) $Node(p'_s)$ is a child of $Node(p_s)$; 2) $\neg\exists Node(\hat{p_s}) \in \mathcal{B}(p_d)$, where $Node(\hat{p_s})$ is an ancestor of $Node(p'_s)$ and a child of $Node(p_s)$.*

Intuitively, the domain of a black node is the largest sub-tree that roots at itself and does not contain any other black nodes. For example, in Fig. 8, the domain of $Node(****)$ is $\mathcal{D}(101^{*}, ****) = \{Node(****), Node(100^{*})\}$.

**Lemma 1.** *For* Action$(p_d, p_s, a)$*, changing cell set $\{(p_d, p'_s) | Node(p'_s) \in \mathcal{D}(p_d, p_s)\}$ to index of $a$ is the minimum.*

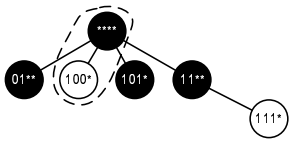



Figure 8: Color tree $CT(101^{*})$ for Fig. 3

From Lemma 1, we know that it is enough to find the domain that need to be updated. We develop update algorithms using depth-first search on the domain. The details of the algorithms is in [47]. Note that the complexity of the algorithms is the size of the domain to be updated. Our algorithms can be pipelined. Using dual-port SRAM [29] [3], TD-table update also does not need to interrupt the lookup process.

# 5 Implementation Design

We further improve the memory footprint and update operations for practical situations.

## 5.1 A Non-Homogeneous FIST Structure

We expect that in practice, only a few prefixes, e.g., the prefixes that belong to famous web servers and data centers, have more next hops than the default ones. It is thus wasteful to leave a row for every destination prefix. To become more compatible to the current router structure and further reduce the SRAM space, we divide the forwarding table into two parts. In the first part each prefix points to a row in TD-table, and in the second part each prefix points directly to an index value. For example, in Fig. 3, destination prefix 011* does not need any specific source prefix, so it is stored in the second part.

In our implementation, we logically divide the table into two parts using a indicator bit in destination index. With this structure, Action$(p_d, f_r(p_d))$ does not update a full row in TD-table if $p_d$ only needs default nexthop.

[3] Current dual-port SRAM can resolve the read-write collision, i.e., read during write operation at the same cell [2].

## 5.2 Updates on Source Default Prefix

We find that the updates on default entries of the source table, i.e., Action($p_d$, *, $a$), can cause a large number of rewrites. This is because source default prefix resides at the root node of the color trees. This means that once it is updated, it may cause a lot of subsequent updates. Unfortunately, the default entry changes more frequently than others, because it has to change when connectivity information of $p_d$ changes. For example, nowadays, the update frequency on connectivity information can reach tens of thousands per second [29].

We propose to isolate default entry from the source table. We remove this entry from source table and rather than being matched explicitly when the full wildcard is hit, the default nexthop is matched when none entry in the source table is matched. We put the default entry in destination index. In Fig. 9, we show the transformation.

Note that with this improvement, some cells in TD-table may be empty. For example, after isolating, $node(100^{*})$ does not belong to the domain of any black node in Figure 8, thus cell (101*, 100*) becomes invalid in Fig. 9. When a packet matches an empty cell, the packet will be forwarded to the nexthop of the default entry in the corresponding destination prefix.

After removing the default entries from the source table, updating on the default nexthop of a destination prefix does not influence TD-table. For example, in Fig. 9, Update$(101^{*}, ****, 3)$ only needs to change the default nexthop index in the destination index to be 3. After isolating, we believe the update frequency of TD-table will be low, with the following two facts: 1) the update of non-connectivity rules will be slow, and it does not have to respond instantly to the changes of network topology; 2) most prefixes in the current forwarding tables are near leaf nodes in color trees [6], so we only need to update a few cells during most updates.

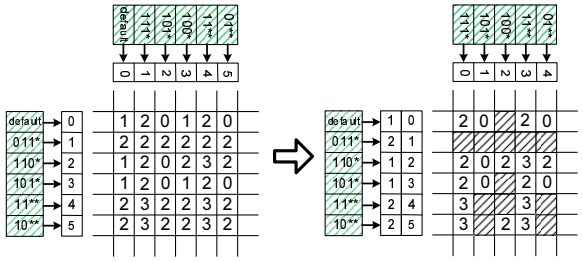



Figure 9: Isolate default entry from source table

## 5.3 Parameter Selection: Fixed Block Deduplication of SRAM Compression

In our fix block deduplication procedure design (Section 3.3.2), two important parameters need to be settled during implementation. We discuss them in detail.

**Bloom filter:** We use the basic bloom filter to accelerate the full storage deduplication process. To avoid un-

necessary lookups in the fingerprint store, there exists a summary vector [48], which uses a vector of $m$ bits. The bloom filter uses $k$ independent hash functions, each mapping a fingerprint randomly to a bit in the summary vector. To keep the false positive rate below 2%, we set $m = 256K$ and $k = 4$.

**Fingerprint store:** The fingerprint store organizes all detected fingerprints. To achieve fast searching, we implement it with 256 buckets. The last byte is used to map each fingerprint to a bucket. We then search in a bucket using binary search.

We set the size of a cell in the catalog table to be 32 bits, the size of a cell in dictionary table to be 8 bits, which is also the size of a TD-cell.

We generate the fingerprints using the SHA-1 function of OpenSSL crypto library [3]. It can process over 2.5Gb SRAM per second when a sub-row has 500 cells. It can even be accelerated by 6+ times if implemented in hardware [16]. Because deduplication overheads are mostly due to computations of fingerprints [31], so 1Gb SRAM can be deduplicated within about 1 second.

# 6 Implementation

We implement the TwoD routing table on a commercial router, Bit-Engine 12004, which supports 4 linecards. In each linecard, there are a CPU board (BitWay CPU8240 with clock rate 100MHz), two TCAM chips (IDT 75K62100), an FPGA chip (Altera EP1S25-780), and several cascaded SRAM chips (IDT 71T75602). Inside the FPGA chip, there exists internal SRAM memory.

Our implementation is based on existing hardware, and does not need any new hardware. We re-design the original destination-based router through rewriting about 1500 lines of VHDL codes with some additional C codes.

The logical implementation is shown in Fig. 10 and a picture of our hardware is shown in Fig. 11.

In data plane, due to our resource limit, we place destination and source tables in different blocks of one TCAM chip. Consequently, the FPGA has to access the TCAM twice for one lookup. Clearly this increases per lookup delay. Note that many hardware support multiple lookups in parallel, e.g., Cisco's TCAM4 that allows four in parallel [23]. The TCAM memory is structured according to L-algorithm [33]. More specifically, prefixes of the same length are clustered together and free space between different clusters is reserved to guarantee fast updates in TCAM.

In Fig. 10, the packet first arrives at the Interface module. After matching, the TCAM module will output the matched prefix, and through the TCAM associated SRAM, FPGA will get the destination and source indexes. Then FPGA accesses the internal SRAM block for the TD-cell. After obtaining the nexthop index, FPGA accesses the mapping table, which resides in another internal SRAM block. Then FPGA gets the next hop information, and delivers the packet to the next processing module - switch co-process module, which will switch the packet to the right interface.

# 7 Performance Evaluation and Simulation

We evaluate our TwoD router through experiments using real data sets. We also conduct simulations to assist evaluation of SRAM compression under various settings. Due to page limitation, our simulation results are in [47].

## 7.1 Evaluation Environment

Our evaluation environment is shown in Fig. 12. There are three components: 1) a PC host with a CPU of Intel Core2 Duo T6570 acting as the control plane, 2) a 4GE linecard equipped with both ACL-like and FIST structures for TwoD router, and 3) a traffic generator (IXIA 1600T) with speed of 4Gbps. The traffic generator is connected to the linecard through optical fibers and the linecard is connected to the PC host through serial cables. The traffic generator sends packets of 64 bytes (including 18 bytes Ethernet Header) at full 4Gbps speeds. The linecard receives the packets, performs lookups and sends the packets back to the traffic generator.

We evaluate the storage footprint for TCAM and SRAM, and the lookup and update processing speed.

## 7.2 Data Sets

We study two practical scenarios: policy routing and load balancing. Both scenarios are based on true demands of CERNET2, which provides access services for universities/institutions in more than 22 major cities of China.

Within each scenario, we generate data sets of 1) TwoD rules that need to be stored in the forwarding table, 2) packets flows for lookup and 3) update sequence.

### 7.2.1 Scenario 1: Policy Routing in CERNET2

CERNET2 has two international exchange centers connecting to the Internet: Beijing (CNGI-6IX) and Shanghai (CNGI-SHIX). During operations, we found that CNGI-6IX is very congested with an average throughput of 1.18Gbps (February 2011); and CNGI-SHIX is much more spared with a maximal throughput of 8.3Mbps at the same time. CERNET2 wants to divert out-going International traffic to CNGI-SHIX (Shanghai portal).

We collect the prefix and FIB information from CERNET2. There are 6973 prefixes in the FIB, and 6406 are foreign prefixes. At the initial stage, we select three universities: Tsinghua University (in Beijing), HUST (in Wuhan) and SCUT (in Guangzhou) to forward their traffic to CNGI-SHIX. We thus simulate three FIBs on three routers, Beijing, Wuhan and Guangzhou (we call each FIB PR-BJ, PR-WH, and PR-GZ).

The traffic flow is generated as follows: Using the real traffic data of the related router, we choose 255 unique

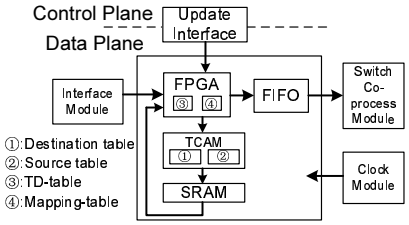



Figure 10: The framework of router design

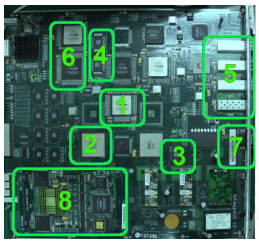



Figure 11: Implementation of FIST on the linecard of Bit-Engine 12004

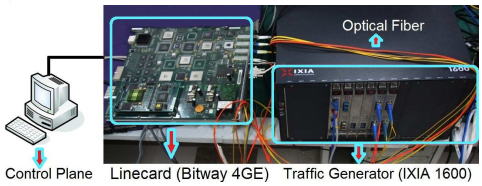



Figure 12: Evaluation environment

*macro flows*, which are identified by their source and destination prefixes. They dominate over 85% of the total traffic. For each macro flow, we obtain its flow rate, and scale it such that the total rate is 4Gbps. We then generate 255 unique packet streams (upper limit of IXIA 1600T), each bound with a macro flow. The IP addresses of each packet are randomly distributed in the address spaces of the prefixes of the related macro flow.

We generate the update sequence on the router of Wuhan as follows: the initial forwarding table only contains destination prefixes, and we add all rules into the forwarding table all at once. In this way, we simulate a common scenario, where ISPs decide to carry out a policy at some time point.

| | PR-BJ | PR-GZ | PR-WH | LB-MO | LB-AF | LB-NI |
|---|---|---|---|---|---|---|
| Rules # | 250366 | 186306 | 365674 | 7118 | 7342 | 7410 |
| Updates # | / | / | 365674 | / | / | 475773 |

Table 2: Data sets overview

### 7.2.2 Scenario 2: Load Balancing in CERNET2

Fig. 13 (Y-axis is anonymized) shows the bandwidth utilization of CNGI-6IX and CNGI-SHIX. The traffic is quite dynamic, thus policy routing itself may not fully solve the load unbalance problem of CERNET2, though it can give higher priority to certain prefixes like Tsinghua University. To have a more thorough solution, we need dynamic load balancing mechanisms in future. As a case study, we collect one Tera-Bytes of NetFlow traffic data during Jan, 2012 on routers of Beijing, Shanghai and Wuhan. We will redistribute the traffic flows.

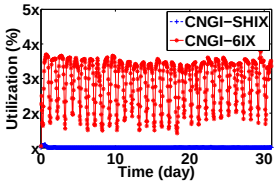



Figure 13: Utilization of CNGI-6IX and CNGI-SHIX

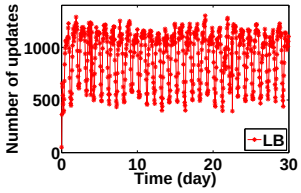



Figure 14: Number of updates for load balancing

We try to redistribute each macro flow to different exchange centers, such that load is optimally balanced. The problem can be reduced to Multi-Processor Scheduling problem [7] which is NP-hard. Thus we use the greedy first-fit algorithm, which greedily assigns each macro flow to the least utilized exchange center. The algorithm achieves an approximation factor of 2.

We construct three forwarding tables, each at different time points, i.e., 6:00, 14:00 and 22:00 during Jan 15, 2012 on the router of Wuhan (we call each forwarding table LB-MO, LB-AF, and LB-EV). Among them, LB-EV is the largest one, because more traffic should be moved when 22:00 is the peak traffic point during a day.

The traffic flow is generated using the same way as Scenario 1. The update sequence is generated as follows: we compute a new load balancing forwarding table every hour, and compare it with that of the previous hour. According to the difference between them, we obtain the update sequence of each hour. Fig. 14 shows the number of updates per hour in this scenario.

Table 2 summarize the number of rules and updates (PR: policy routing LB: load balancing). For comparison, we use the ACL-like structure as a benchmark.

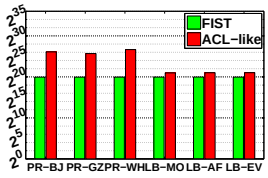
## 7.3 Evaluation Results

### 7.3.1 Storage of Basic FIST Structure

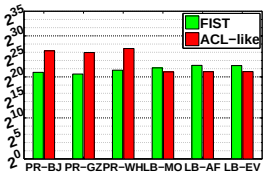
In Fig. 15(a), we can see that the TCAM space in FIST can be 1/50 as compared to ACL-like structure. For example, in PR-WH, FIST consumes 1Mb TCAM storage, while ACL-like structure consumes more than 72Mb. In the LB scenarios, FIST gain is smaller. This is because in the PR scenario, many rules share the same destination or source prefixes yet in the LB scenario, there are much less two dimensional rules.

In Fig. 15(b), we can see that, in the PR scenario, the SRAM space in FIST can be 1/40 as compared to ACL-like structure. For example, in PR-WH, FIST consumes 3Mb SRAM storage, while ACL-like structure consumes 125Mb. In the LB scenario, FIST consumes more SRAM storage. This is because in the PR scenario, although there are many rules, the TD-table is very dense, and nexthop index further condenses the nexthop information. In the LB scenario, the TD-table is much sparser. In Section 7.3.3, we show that the SRAM storage of FIST can be greatly reduced, due to the flexibility of SRAM.

To give an overview on the benefits we can achieve from FIST, we estimate the cost and power consumption of FIST and ACL-like structures. We set the price, and power consumption of one MB SRAM to be $27 and




(a) TCAM storage space




(b) SRAM storage space

Figure 15: Size of each forwarding table with basic structure

0.12 Watt, one MB TCAM to be $200 and 15 Watt [30]. We use the simple linear model to estimate them [23]. In Table. 3, we can see that the cost and power consumption of FIST are far less than those of ACL-like structure. For example, to accommodate PR-WH, the FIST costs $36.2 and consumes 1.94 Watts while ACL-like structure costs $3057.4 and consumes 199.35 Watts. This is because FIST moves the storage from expensive and high power consuming TCAM to SRAM.

| | Cost($) | | Power (Watt) | |
|---|---|---|---|---|
| | FIST | ACL-like | FIST | ACL-like |
| PR-BJ | 32.7 | 2093.3 | 1.92 | 136.49 |
| PR-GZ | 30.8 | 1557.7 | 1.91 | 101.57 |
| PR-WH | 36.2 | 3057.4 | 1.94 | 199.35 |
| LB-MO | 41.5 | 59.5 | 1.97 | 3.88 |
| LM-AF | 50.3 | 61.4 | 2.02 | 4.00 |
| LM-EV | 49.3 | 62.0 | 2.01 | 4.04 |

Table 3: Estimated price and power consumption of each forwarding table

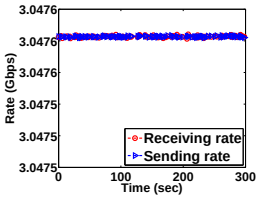



Figure 16: Lookup speed without update

### 7.3.2 Lookup and Update Operations

**Lookup Speed:** In Fig. 16, we show the lookup speed without updates. We can see that without updates, both sending and receiving rates reach line speeds (Ethernet frame contains 8 bytes preamble and 12 bytes gap, thus the maximum rate is $4 \times \frac{64}{64+20} \approx 3.0476$Gbps). We also look into the data traces, and find none packet loss. Note that the speed reaches the upper limit of the linecard we use, and it can be higher with better linecards.

**TCAM Accesses During Update:** We evaluate the number of accesses to TCAM because updates in TCAM will interrupt the lookup. In Fig. 17, we show the number of TCAM accesses per 100 updates in PR and LB scenarios. We can see that the number of TCAM accesses that FIST causes can be three orders of magnitude less than ACL-like structure. For example, in the PR scenario, FIST causes 2-3 TCAM accesses per 100 updates while ACL-like structure causes several thousands. This is because FIST stores much less information in TCAM.

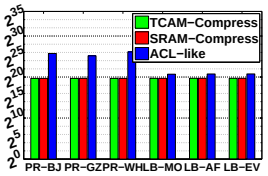
In Fig. 18, we show the lookup speeds of FIST, with different update frequencies, i.e., 500, 5000, and 50000 updates/sec, during 5 minutes. In Fig. 18(a), we can see that in the PR scenario, FIST has no influence on lookup while ACL-like structure degrades the lookup speeds by 7% in the worst case. This is because FIST causes much less accesses to TCAM. In Fig. 18(b), we can see that in the LB scenario, FIST does influence the lookup speeds when there are 50,000 updates per second, however, the influence is still much smaller than ACL-like structure.

We conclude that FIST structure will not impose high update burden on lookup. In the PR scenario, all updates can be finished in less than 10 seconds without influencing lookup, which is fast enough for installing a policy. In the LB scenario, the maximum number of updates per hour is 1,301, that can be finished within 1 second without influencing lookup.

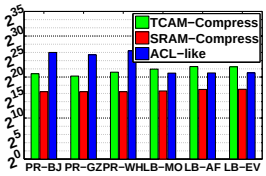
**SRAM Accesses During Update:** FIST causes more accesses to SRAM. Although it does not interrupt the lookup with dual-port SRAM, it is limited by hardware capacities and computing resources. In Fig. 19, we show the number of accesses to SRAM per 100 updates. We can see that the number of SRAM accesses in FIST is one order less than ACL-like structure. For example, in the PR scenario, each update causes only 1 SRAM access in FIST, and tens of SRAM accesses in ACL-like structure. This is because in FIST, an update only needs to rewrite part of the conflicted TD-cells. In ACL-like structure, frequent moves in TCAM lead to subsequent moves in SRAM.

### 7.3.3 Storage of FIST with Improvements

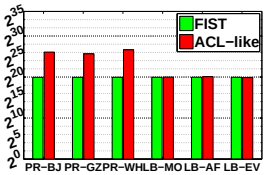
Except the basic setting, we evaluate in other two settings – after compression, with non-homogeneous structure.



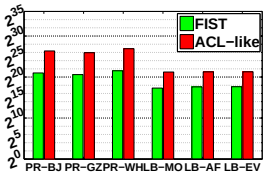

(a) TCAM-compression



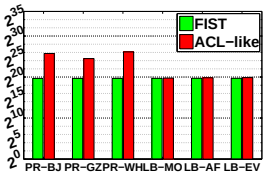

(b) SRAM-compression



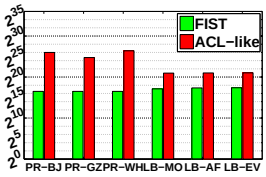

(c) TCAM-non-homogeneous




(d) SRAM-non-homogeneous




(e) TCAM-compression&non-homo




(f) SRAM-compression&non-homo

Figure 20: Size of each forwarding table

**Compression:** In Fig. 20(a), we show the TCAM space after compression. We also compress ACL-like structure by minimizing the number of TowD rules. We can see that, after TCAM compression, FIST still consumes much less TCAM storage than ACL-like structure. In Fig. 20(b), we show the SRAM space after compression. We can see that FIST can further compress SRAM after TCAM compression. For example, SRAM storage

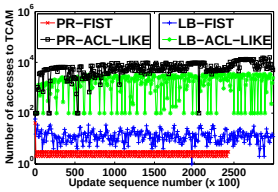
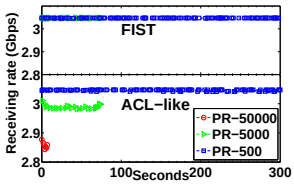
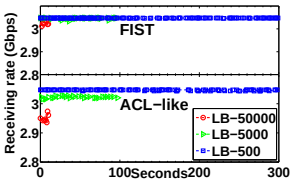



Figure 17: TCAM accesses




(a) Policy routing scenario




(b) Load balancing scenario

Figure 18: Lookup speeds with updates

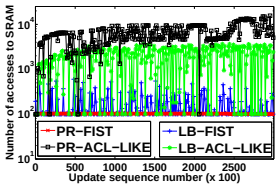



Figure 19: SRAM accesses

of PR-WH is compressed to be less than 90K bits. This is because 1) the flexible mapping structure of FIST; 2) data redundancies in TD-table. However, in the ACL-like forwarding tables, the SRAM storage is proportional to the TCAM storage, and can not be further compressed.

**Non-Homogeneous Structure:** In Fig. 20(c), we show the TCAM space with non-homogeneous structure. Non-homogeneous structure does not save TCAM storage in FIST but saves TCAM storage in ACL-like structure. However, to support non-homogeneous structure, ACL-like structure must be physically divided, because most TCAM chips only support uniform entry width. In contrast, with FIST, we can flexibility and logically divide the table into two parts. In Fig. 20(d), we show the SRAM space with non-homogeneous structure. We can see that, with non-homogeneous structure, FIST consumes much less SRAM storage than ACL-like structure in all forwarding tables.

**Combine Non-Homogeneous Structure with Compression:** In Fig. 20(e) and 20(f), we apply non-homogeneous structure and compression techniques to all forwarding tables. We can see that TCAM and SRAM spaces in FIST are much less than ACL-like structure.

### 7.3.4 Evaluation Summary

Our evaluation shows that FIST can well accommodate 365674 rules (PR-WH). For all scenarios, FIST consumes at most 1Mb TCAM and 7Mb SRAM space. Such excellent performance is because we fully eliminate the multiplicative effect in TCAM and we can take advantage of the flexibility of SRAM to design advanced compression schemes. As a matter of fact, we believe that our limited scenarios from CERNET2 cannot fully illustrate the potential of FIST. Our experience shows FIST can easily accommodate more than 1 million TwoD rules. Our TwoD router achieves constant lookup time and works well with 50000 updates per second and we believe these do not hit the limit of FIST potential as well.

## 8 Discussion and Current Limitations

The current TCAM chip in market has a storage space of 36Mb, and SRAM chip in the market has 144Mb (72Mb TCAM and 288Mb SRAM are on the roadmap of major vendors) [14]. Other memory products such as RLDRAM can provide similar performance to SRAM, having 16 bytes reading with random access time of 15 ns with memory denominations of 576 Mbits/chip [15]. Multiple chips can be used, e.g., linecards of Bit-Engine 12004 support 4 SRAM chips. The Juniper MX960 is estimated to use 32 SRAM chips in 2007 [4].

The current number of destination prefixes in backbone ISP routers is 400,000. Given IPv4, this transfers into $400,000 \times 36 = 14.4$Mb (the common TCAM width for IPv4 is 36 bits). We expect that source prefixes (representing rules) are much smaller than destination prefixes. Suppose that there are 10,000 source prefixes, which transfer into 0.36Mb. Overall, this scenario requires 14.4 + 0.36 = 14.76Mb TCAM and $400,000 \times 10,000 \times 8 = 32$Gb SRAM. We can see that TCAM is far enough. Yet for SRAM, assume there is a FIST compression ratio of one-third, and 8 or 12 SRAM chips are used (capable for most routers nowadays), there is still a gap around a factor of 4. Note that 400,000 destination prefixes and 10,000 source prefixes can mean millions of rules.

For mid or small scale ISPs such as CERNET2, the TwoD router is already applicable. We admit that there is still storage limitation for large scale ISPs. Note that this limitation is on SRAM, not TCAM. As SRAM is flexible, we believe better data structure and compression schemes can be developed in future studies. From past experience in memory/storage development, and the fact that SRAM cheaper so it may be easier to add more chips, we are optimistic that such gap can be overcame.

## 9 Conclusion

We presented a design and implementation of a two dimensional router, where forwarding decisions are based on both destination and source addresses. The motivation was to expand routing semantics to support increasing demands of differentiate services from ISPs like CERNET2. Adding source prefixes into FIB leads to TCAM storage explosion. We thus made a neat separation of TCAM and SRAM, where we maintained the fast lookup of TCAM and took advantage of the large storage space/flexibility of SRAM. We proved the correctness of our design and proposed algorithms for further TCAM/SRAM compression. We developed algorithms for routing operations of lookup and updates. We implemented the TwoD router on the linecard of a commercial router. Our design does not need any new devices. We conducted comprehensive experiments and simulations with the real data sets from CERNET2. The results showed that our TwoD router can be used for ISPs like CERNET2 to support practical services such as policy routing or load balancing.

## References


[1] Bgp routing table analysis reports. `http://bgp.potaroo.net`.

[2] Cyclone handbook. `www.altera.com/literature/hb/cyc/cyc_c51007.pdf`.

[3] Openssl. `http://www.openssl.org`.

[4] Router fib technology. `http://www.firstpr.com.au/ip/sramip-forwarding/router-fib/`.

[5] F. Baboescu, P. Warkhede, S. Suri, and G. Varghese. Fast packet classification for two-dimensional conflict-free filters. *Comput. Netw.*, 50(11):1831–1842, 2006.

[6] A. Basu and G. Narlikar. Fast incremental updates for pipelined forwarding engines. *IEEE/ACM Trans. Netw.*, 13:690–703, 2005.

[7] J. Blazewicz, M. Drabowski, and J. Weglarz. Scheduling multiprocessor tasks to minimize schedule length. *IEEE Trans. Comput.*, 35(5):389–393, 1986.

[8] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, pages 636–646, 2003.

[9] Y. Chiba, Y. Shinohara, and H. Shimonishi. Source flow: handling millions of flows on flow-based nodes. In *Proc. ACM SIGCOMM'10*, New Delhi, India, Sep 2010.

[10] R. P. Draves, C. King, S. Venkatachary, and B. N. Zill. Constructing optimal ip routing tables. In *Proc. IEEE INFOCOM'99*, New York, NY, March 1999.

[11] P. Gupta. *Algorithms for Routing Lookups and Packet Classification*. PhD thesis, Stanford University, Dec 2000.

[12] P. Gupta and N. McKeown. Packet classification on multiple fields. In *Proc. ACM SIGCOMM'99*, Cambridge, Massachusetts, United States, Aug 1999.

[13] P. Gupta and N. McKeown. Algorithms for packet classification. *Network, IEEE*, 15(2):24 –32, 2001.

[14] C. Hermsmeyer, H. Song, R. Schlenk, R. Gemelli, and S. Bunse. Towards 100g packet processing: Challenges and technologies. *Bell Lab. Tech. J.*, 14(2):57–79, 2009.

[15] K.Fall, G.Iannaccone, S.Ratnasamy, and P.Godfrey. Routing tables: Is smaller really much better? *BT Technology Journal*, 24:119–129, 2006.

[16] M. Khalil-Hani, V. P. Nambiar, and M. N. Marsono. Hardware acceleration of openssl cryptographic functions for high-performance internet security. In *Proc. International Conference on Intelligent Systems, Modelling and Simulation*, Liverpool, UK, Jan 2010.

[17] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee. Internet traffic classification demystified: myths, caveats, and the best practices. In *Porc. ACM CoNEXT'08*, Madrid, Spain, Dec 2008.

[18] J. Kim, M.-C. Ko, H.-K. Kang, and J. Kim. A hybrid ip forwarding engine with high performance and low power. In *Proc. ICCSA'09*, Seoul, Korea, Jun 2009.

[19] T. V. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. *SIGCOMM Comput. Commun. Rev.*, 28(4):203–214, 1998.

[20] C.-L. Lee and P.-C. Wang. Scalable packet classification by tcam entry encryption algorithm. *J. High Speed Netw.*, 16(3):275–283, 2007.

[21] A. Liu, C. Meiners, and E. Torng. Tcam razor: A systematic approach towards minimizing packet classifiers in tcams. *Networking, IEEE/ACM Transactions on*, 18(2):490 –500, 2010.

[22] H. Lu and S. Sahni. Conflict detection and resolution in two-dimensional prefix router tables. *IEEE/ACM Trans. Netw.*, 13(6):1353–1363, 2005.

[23] Y. Ma and S. Banerjee. A smart pre-classifier to reduce power consumption of tcams for multi-dimensional packet classification. In *Proc ACM SIGCOMM'12*, Helsinki, Finland, Aug 2012.

[24] N. Mandagere, P. Zhou, M. A. Smith, and S. Uttamchandani. Demystifying data deduplication. In *Proc. ACM/IFIP/USENIX Companion'08*, Leuven, Belgium, Dec 2008.

[25] C. Meiners, A. Liu, and E. Torng. Bit weaving: A non-prefix approach to compressing packet classifiers in tcams. In *Proc. IEEE ICNP'09*, Orlando, Florida, Oct 2009.

[26] C. Meiners, A. Liu, and E. Torng. *Hardware Based Packet Classification for High Speed Internet Routers*. Springer, 2010.

[27] C. R. Meiners, A. X. Liu, E. Torng, and J. Patel. Split: Optimizing space, power, and throughput for tcam-based classification. In *Proc. ACM/IEEE ANCS'11*, Brooklyn, NY, Oct 2011.

[28] C. R. Meiners, J. Patel, E. Norige, E. Torng, and A. X. Liu. Fast regular expression matching using small tcams for network intrusion detection and prevention systems. In *Proc. USENIX Security'10*, Washington, DC, Aug 2010.

[29] T. Mishra and S. Sahni. Duos - simple dual tcam architecture for routing tables with incremental update. In *Proc. IEEE ISCC'10*, Riccione, Italy, Jun 2010.

[30] D. Perino and M. Varvello. A reality check for content centric networking. In *Proc. ACM SIGCOMM Workshop ICN'11*, Toronto, Ontario, Canada, Aug 2011.

[31] C. Policroniades and I. Pratt. Alternatives for detecting redundancy in storage systems data. In *Proc. USENIX ATEC'04*, Jun 2004.

[32] S. Quinlan and S. Dorward. Venti: A new approach to archival data storage. In *Proc. USENIX FAST'02*, Monterey, CA, Jan 2002.

[33] D. Shah and P. Gupta. Fast updating algorithms for tcams. *IEEE Micro*, 21(1):36–47, 2001.

[34] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *Proc. ACM SIGCOMM'03*, Karlsruhe, Germany, Aug 2003.

[35] H. Song, J. Turner, and S. Dharmapurikar. Packet classification using coarse-grained tuple spaces. In *Proc. ACM/IEEE ANCS'06*, San Jose, California, USA, Dec 2006.

[36] V. Srinivasan, S. Suri, and G. Varghese. Packet classification using tuple space search. In *Proc. ACM SIGCOMM'99*, Cambridge, Massachusetts, United States, Aug 1999.

[37] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. In *Proc. ACM SIGCOMM'98*, Vancouver, British Columbia, Canada, Aug 1998.

[38] S. Suri, T. Sandholm, and P. Warkhede. Compressing two-dimensional routing tables. *Algorithmica*, 35:287–300, 2003.

[39] D. Taylor and J. Turner. Scalable packet classification using distributed crossproducing of field labels. In *Proc. IEEE INFOCOM'05*, Miami, FL, Mar 2005.

[40] D. E. Taylor. Survey and taxonomy of packet classification techniques. *ACM Comput. Surv.*, 37(3):238–275, 2005.

[41] D. E. Taylor and J. S. Turner. Classbench: a packet classification benchmark. *IEEE/ACM Trans. Netw.*, 15(3):499–511, 2007.

[42] B. Vamanan, G. Voskuilen, and T. N. Vijaykumar. Efficuts: optimizing packet classification for memory and throughput. In *Proc. ACM SIGCOM'10*, New Delhi, India, Aug 2010.

[43] J. van Lunteren and T. Engbersen. Fast and scalable packet classification. *Selected Areas in Communications, IEEE Journal on*, 21(4):560 – 571, 2003.

[44] J. Wu, J. Bi, M. Bagnulo, F. Baker, and C. Vogt. Source address validation improvement framework. Internet Draft, Mar 2011. draft-ietf-savi-framework-04.txt.

[45] M. Xu, J. Wu, S. Yang, and D. Wang. Two dimensional ip routing architecture. Internet Draft, Mar 2012. draft-xu-rtgwg-twod-ip-routing-00.txt.

[46] M. Xu, S. Yang, D. Wang, and J. Wu. Two dimensional-ip routing. In *Proc. IEEE ICNC'13*, San Diego, USA, Jan 2013.

[47] S. Yang, D. Wang, M. Xu, and J. Wu. Two dimensional router: Design and implementation. Technical report, Tsinghua University, Aug 2012. `http://www.wdklife.com/tech.pdf`.

[48] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proc. USENIX FAST'08*, San Jose, California, Feb 2008.