Scalable Forwarding Tables for Supporting Flexible Policies in Enterprise Networks

Shu Yang*†, Mingwei Xu*†, Dan Wang‡§, Gautier Bayzelon*†, Jianping Wu*†
*Department of Computer Science and Technology, Tsinghua University
†Tsinghua National Laboratory for Information Science and Technology
‡Department of Computing, The Hong Kong Polytechnic University
§The Hong Kong Polytechnic University Shenzhen Research Institute

***Abstract*—With increasing demands for more flexible services, the routing policies in enterprise network becomes much richer. This has placed a heavy burden to the current router forwarding plane to support the increasing number of policies, primarily due to the limited capacity in TCAM. This hinders the development of new network services.**

In this paper, we present the design and implementation of a new forwarding table structure. It separates the functions of TCAM and SRAM and maximally utilizes the large & flexible SRAM. We progressively design a set of schemes, to maintain correctness, compress storage, and achieve line-card speeds. We also design incremental update algorithms that bring less accesses to memory. We present implementation designs and evaluate our scheme with a real implementation on a commercial router using real data sets. Our design does not require new devices. The evaluation results show that the performance of our forwarding tables is promising.

I. Introduction

Enterprise networks have attracted great attentions along the increasing demands from more flexible policies in enterprises [22]. A typical enterprise network hosts thousands of routers [20]. Network management is responsible for 80% of the IT budget and 62% of the outages [27]. Different from the backbone networks, which mainly provide reachability services, enterprise networks need to support much more fine-grained routing policies. For example, 1) **Access Control:** An hospital should implement strict access control to gurantee the security of patients' electronic records; 2) **Performance:** A bank should choose secure paths for its bank applications, and low-latency paths for its financial applications [27]. Other example include load balancing, network virtualization, etc.

There are many solutions to support these policies. For example, policy-based routing (PBR) [5] installs policies into access control list (ACL), and multi-topology routing (MTR) [8] supports multiple independent control and forwarding planes. Currently, engineers in IETF are proposing traffic-class routing (TCR) [14][10] that adds more information, e.g., source address, into routing, such that routing decisions can be made based on both destination and source addresses.

Although these solutions differ greatly in control plane; they all need an enhanced forwarding plane to support large number of forwarding rules based on an increasing number of routing policies. Nevertheless, current solutions are not scalable. For example, MTR uses a separate forwarding table for each topology, but it can only support a limited number (32 in most cases [8]) of topologies while current enterprise networks require more [3]; TCR recommends using one forwarding table per source prefix. This scales even worse than MTR and is only suitable for small networks.

Many other solutions, like PBR uses the traditional cisco Access Control List (ACL) structure (we call it ACL-like structure thereafter) in TCAM. We show a typical forwarding table with ACL-like structure in Table. I. For illustration purposes, we use 4-bit IP addresses. Here the destination and source prefixes are concatenated as an entry in TCAM. When a packet with destination address of 1011 and source address of 1111 arrives, it will match destination prefix 101* and source prefix 11** according to the longest match first (LMF) rule; and then forward the packet to the interface of 1.0.0.2. This 'fat' TCAM structure provides a fast lookup speed. However, using ACL-like structure means the Forwarding Information Base (FIB) table changes from $\{\text{destination}\} \rightarrow \{\text{action}\}$ to $\{(\text{destination, source})\} \rightarrow \{\text{action}\}$. This structure tremendously increases the TCAM resources. In the worst case, the number of TCAM entries can be $O(N \times M)$, where N and M are the size of destination and source addresses.

China Education and Research Network-2 (CERNET2) is currently using this ACL-like structure. CERNET2 wants to carry out policy routing between about 6,000 destination prefixes and 100 source prefixes. This results in a requirement of at least 600,000 entries in TCAM. Many modern enterprise networks face similar problems after widely deploying lots of security, QoS and privacy functions in networks [3]. It is widely known that TCAM is scarce in resource due to its small storage size, high cost and high power consumption. It is also difficult to compress it due to its unique structure [15].

The research is supported by the National Basic Research Program of China (973 Program) under Grant 2012CB315803, the National Natural Science Foundation of China (61073166 and 61133015),the National High-Tech Research and Development Program of China (863 Program) under Grants 2011AA01A101. Dan Wang's work is supported in part by National Natural Science Foundation of China (No. 61272464), RGC/GRF PolyU 5264/13E, HK PolyU 1-ZVC2, G-UB72.

978-1-4799-3360-0/14/$31.00

Thousands of rules in ACL will bring large overheads to an enterprise [13]. However, existing forwarding plane solutions do not scale well in TCAM, and this has become a bottleneck for developing new services in enterprises [7][16].

#	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 I: A Two Dimensional Forwarding Example

In this paper, we present the design and implementation of a scalable forwarding table, which can handle expanded policies in enterprise networks. Our new forwarding table structure is called FISE (**FI**B **S**tructure for **E**nterprise). The key idea of FISE is to move the redundancies from TCAM to flexible, cheap and power-saving SRAM. We show an example of FISE in Fig. 1. FISE stores destination and source prefixes in two separate TCAM tables, and store other information in SRAM. In Table. I, we only need to store destination prefixes ****, 011*, 110*, 101*, 11** and 10** in one TCAM table, and source prefixes ****, 101*, 11**, 01**, 111* and 100* in another TCAM table. Thus, we substantially reduce the TCAM storage space; there are at most $N+M$ TCAM entries. Besides, SRAM is much more flexible and we can develop a dozen more techniques to eliminate the redundancies.

However, with FISE, there are many challenges: 1) we need to guarantee the line-card speeds and correctness of packet forwarding, and we will show that this is not straightforward (Section II); 2) in principle, an update in FISE may incur multiple accesses in memory, we need to develop incremental update algorithms to minimize such accesses (Section III); and 3) in practice, we want to make FISE more scalable and accommodate a number of rules in the order of millions, which is 100 times of that of today (Section IV). In this paper, we present a comprehensive study on FISE and progressively solved all these problems. We implement the FISE on a commercial router, Bit-Engine 12004 and we present a set of practical implementation designs (Section V). Note that through careful redesign of the hardware logic, FISE does not require new devices. We carry out comprehensive evaluations with the real implementation, using the real topology, FIB, prefix and traffic data from CERNET2 (Section VII). The results show that FISE can achieve line-card speed, save TCAM and SRAM storage, and bring acceptable update burden.

II. FISE Structure and Lookup

A. The Matching Rule

We first present the definition of the matching rule. Let d and s denote the destination and source addresses, p_d and p_s denote the destination and source prefixes. Let a denote an action, more specifically, the next hop. The entries of the storage should be 3-tuples (p_d, p_s, a).

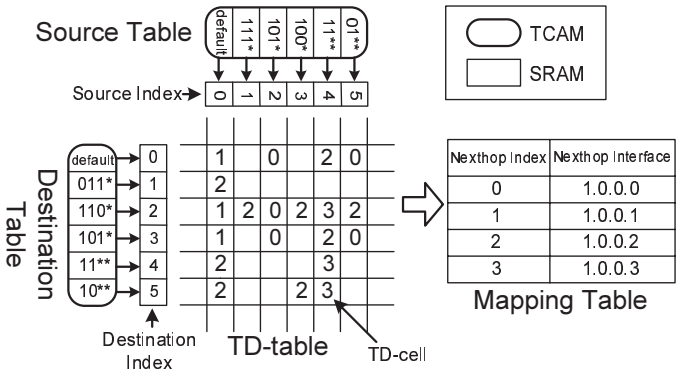


Fig. 1: FISE: A forwarding table structure for enterprise networks

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

We match destination prefixes first, rather than match destination and source prefixes with the same priority [12], to guarantee reachability and avoid routing loops in layer-3 [4].

B. FISE Design Details

1) FISE Basics: The new structure FISE has two tables in TCAM and another two tables in SRAM (see Fig. 1). 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 **T**wo **D**imensional 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. The (p_d, p_s) cell in the TD-table stores an index value. From this index value, a is stored in the corresponding position of the mapping table. We store the index value rather than the next hop a in the TD-table, because the next hop information is much longer.

We show an example in Fig. 1. 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 $(110*, 11**)$ in 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.

Theorem 1: The TCAM storage space of FISE is $O(N+M)$ bits. The SRAM storage space of FISE is $O(N \times M)$ bits.

Proof: The destination table has N entries, the source table has M entries, thus TCAM space is $O(N+M)$ bits. TD-table dominates the SRAM, and has $O(N \times M)$ cells. ■

Clearly, FISE 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 [11][1]. Moreover, SRAM is more flexible than TCAM, thus we can develop kinds of techniques to eliminate the redundancies in SRAM.

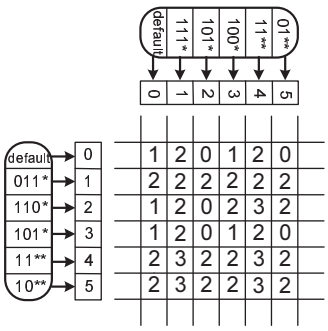


Fig. 2: Apply TD-Saturation()

2) TD-cell Saturation: We establish a TD-table by inserting the entries into an empty TD-table. Fig. 1 shows the TD-table after inserting the entries in Table. I.

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 Definition 1, 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 a 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 values. As an example, using this algorithm, the TD-table of Fig. 1 becomes Fig. 2.

Theorem 2: FISE (with TD-Saturation()) correctly handles the matching rule defined in Definition 1.

Proof: Suppose not. The packet will match another rule other than $(\hat{p_s}, \hat{p_d}, \hat{a})$. If the rule does not belong to $\mathcal{S}$, then p_d is not matched. If the rule does not belong to $\mathcal{S}'$, then p_s should not have been matched. If the rule is not $(\hat{p_s}, \hat{p_d}, \hat{a})$, then p_s is not the LMF match given p_d is matched. ■

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

```
1 begin
    // R is the set of forwarding rules to be stored
2   foreach p_d, p_s do
3     if ∄(p_d, p_s, a) ∈ R then
4       S = {(p̄_s, p̄_d, ā) ∈ R | p̄_d = p_d}
5       S' = {(p̃_s, p̃_d, ã) ∈ S | p̃_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.
8 end
```

C. A Non-Homogeneous FISE Structure

We expect that in practice, only a few prefixes, e.g., the prefixes that belong to private information, 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 FIB 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. 1, 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 an indicator bit in the destination index. We illustrate more details in Section V.

D. FISE Lookup

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

The lookup action is shown in Fig. 3. When a packet arrives, FISE 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 a FIFO buffer, which resolves the un-matching clock-rates between TCAM and SRAM. FISE then obtains the destination index and source index. FISE can thus identify the cell in the TD-table, and return the index value. Using this index, FISE looks up the mapping table, and returns the nexthop.

Theorem 3: The lookup speed of FISE is one TCAM plus three SRAM clock cycles.

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

As a comparison, the conventional forwarding table stores prefixes in one TCAM, and accesses both TCAM and SRAM once during a lookup.

We develop a pipeline lookup process (see Fig. 4) 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 FISE can achieve one packet per clock rate.

Observation 1: The lookup speed of the FISE (with pipelining) is the same as conventional forwarding tables.

III. FISE Incremental Update

Although TD-Saturation() guarantees the correctness of FISE. It need to re-compute all conflicted cells, and re-write them in SRAM once update happens. Suppose that there are 10,000 source prefixes, and 500 updates on destination prefixes per second. In the worst case, there are 5,000,000 accesses in SRAM per second, which almost exceeds the speed of hardware (in Bit-Engine 12004, line-cards work at 100MHz, and line-cards need 20 clock cycles for a read/write operation). Note that although update is necessary, not all cells need to be re-written in the update process. In this section, our objective is to find an incremental algorithm that minimizes the number of *cell updates*. We use function $TD(\cdot, \cdot)$ to denote the TD-table. 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

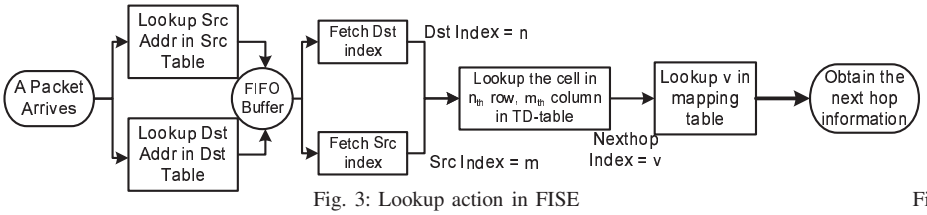


Fig. 3: Lookup action in FISE

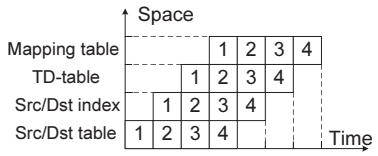


Fig. 4: Space-time diagram of the lookup pipeline

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 first formally present the problem.

Problem 1: **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 the set $\{(p_d, p_s)|TD(f_r(p_d), f_c(p_s)) \neq TD'(f_r(p_d), f_c(p_s))\}$ is minimized.

To achieve this, we will first build a prefix tree to organize the cells. With this tree, we will develop algorithms for insertion and deletion where only part of the cells will be updated. This tree is stored in the control plane (more specifically in DRAM). Note that current routers also store 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 then prove that our algorithms indeed minimize the computation costs and the number of cell rewrites.

A Colored Tree Structure and Update Algorithm

We build a prefix tree called *colored tree*, i.e., $CT(p_d)$, for each destination prefix p_d. The tree includes all source prefixes in the source table as nodes $Node(p_s)$. $Node(p_s)$ is an ancestor of $Node(p'_s)$ if p_s is 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. 5.

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. 5, we show a colored 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 colored trees.

Definition 3: 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 subtree that roots at itself and does not contain any other black nodes. For example, in Fig. 5, the domain of $Node(****)$ is $\mathcal{D}(101*, ****) = \{Node(****), Node(100*)\}$.

Theorem 4: When updating rule (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.

Proof: We prove it by contradiction. Assume another smaller cell set exists, indicating that at least one cell $(p_d, \hat{p_s})$, where $Node(\hat{p_s}) \in |\mathcal{D}(p_d, p_s)$, is not set to the index value of a. Then if a packet matches p_d and $\hat{p_s}$ within FISE, it should match rule (p_d, p_s, a). Thus the index value is wrong. ■

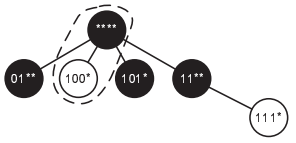


Fig. 5: Colored tree $CT(101*)$ for Fig. 1

Note that the complexity of update algorithms is the size of the domain to be updated. Our algorithms can also be pipelined. Moreover, using dual-port SRAM [18], TD-table update also does not need to interrupt the lookup process.

IV. Practical Considerations

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

A. Compressing FISE

With FISE, we can further minimize both the TCAM and SRAM space. We first formally define equivalence of two tables. As such, we can select another table in the equivalence class that has the minimum size. First, we use a 5-tuple $\{P_d, P_s, f_r(\cdot), f_c(\cdot), TD(\cdot,\cdot)\}$ to denote a FISE table.

Definition 4: $\{P'_d, P'_s, f'_r(\cdot), f'_c(\cdot), TD'(\cdot,\cdot)\}$ 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.

Algorithm 2: Compress-TCAM($P_d, P_s, f_r(\cdot), f_c(\cdot), TD(\cdot,\cdot)$)

```
   Output : {P'_d, P'_s, f'_r(·), f'_c(·), TD'(·,·)}
1  begin
2      Eliminate prefixes that will never be matched
3      ∀p_d ∈ P_d, DF(p_d) = SHA-1(TD(f_r(p_d), ·))
4      {P'_d, DF'(·)} ← ORTC(P_d, DF(·))
5      ∀p'_d ∈ P'_d, f'(p'_d) ← f(p_d), ∃p_d ∈ P_d, DF'(p'_d) = DF(p_d)
6      ∀p_s ∈ P_s, SF(p_s) = SHA-1(TD(·, f_c(p_s)))
7      {P'_s, SF'(·)} ← ORTC(P_s, SF(·))
8      ∀p'_s ∈ P'_s, f'(p'_s) ← f(p_s), ∃p_s ∈ P_s, SF'(p'_s) = SF(p_s)
9      ∀p'_d ∈ P'_d, p'_s ∈ P'_s, TD'(f'_r(p'_d), f'_c(p'_s)) ← TD(f'_r(p'_d), f'_c(p'_s))
10 end
```

1) TCAM Compression: We develop Compress-TCAM(), based on algorithm Optimal Routing Table Constructor (ORTC) [6], which computes the minimized TCAM for a one dimensional forwarding table. In Compress-TCAM(), we first map each row/column vector to a scalar using SHA-1 function[1], then apply ORTC separately to destination and source tables in TCAM.

The complexity of Compress-TCAM() is $O(N \times M)$, because the complexity of Line 4 and 7 is $O(N)$ and $O(M)$, and the complexity of SHA-1 is $O(N \times M)$. However, with hardware support, SHA-1 can process SRAM at 2.5Gb/s.

[1]SHA-1 has much smaller collision probability than hardware error rate.

2) SRAM Compression: Minimizing destination/source tables also reduces SRAM space. However, we can further compress TD-table in SRAM.

To find the minimum TD-table, we simply compress the TD-table by merging duplicated rows (columns). Different with traditional structures, where only aggregation is possible. Due to flexibility in SRAM, FISE indirectly points to the index values through destination/source indexes. If two rows/columns pointed by two prefixes are identical, we can eliminate one of them by making their destination/source indexes point to the same row/column.

Theorem 5: Eliminating the duplicated rows and columns computes the minimum TD-table.

Proof: We give the proof in [26]. ■

With the compression techniques, the update algorithms in Section III remains the same. Besides, we can compress the full table periodically, after multiple updates, rather than every time [24]. Thus, the compression will not influence the FISE performance. Here, SRAM compression only relies on merging. However, we can further compress SRAM due to its flexibility. In [26], we show that we can divide rows into sub-rows, and eliminate duplicated sub-rows even their related rows are not identical.

B. Reducing Update Burden on TD-table

Although the update actions minimize the number of accesses to memory, we find that the updates on default entries of the source table, e.g., Insert(p_d, *, a), can cause a large number of rewrites. This is because 1) source default prefix resides at the root node of the colored trees, thus updating a default entry may cause a lot of subsequent updates; 2) Unfortunately, the default entry changes more frequently than others, because it represents connectivity of destination prefixes. Nowadays, the update frequency on connectivity information can reach tens of thousands per second [18].

We propose to isolate default entries from the source table. We remove these entries from the source table and rather than being matched explicitly when the full wildcard is hit, the default entry is matched when none entry in the source table is matched. In Section V, we will illustrate this in detail.

Note that with this improvement, some cells in TD-table may be empty. This is because in a colored tree after removing the root node, a white node may do not belong to the domain of any black node. For example, in Fig. 5, after isolating $node(****)$, $node(100*)$ does not belong to the domain of any black node, thus cell (101*, 100*) becomes empty. When a packet matches an empty cell, the packet will be forwarded to the nexthop of the default entry.

V. Implementation

As a proof-of-concept, we implement FISE on a commercial router, Bit-Engine 12004, which supports 4 line-cards. Each line-card has a CPU board (clock rate 100MHz), two TCAM chips (IDT 75K62100), an FPGA chip (Altera EP1S25-780), and several cascaded SRAM chips (IDT 71T75602). The FPGA has internal SRAM memory.

Our implementation is based on existed hardware, and does not need any new device. We re-design the hardware logic through rewriting about 1500 lines of VHDL code (not including C code) of the original destination-based version.

A. Router Framework

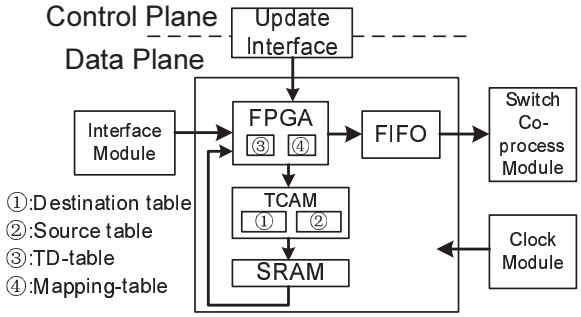
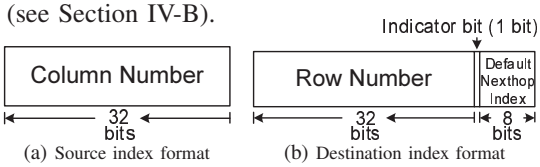


Fig. 6: The framework of router design

In Fig. 6, we show the framework of our design. In data plane, the packet first arrives at the Interface module. After matching, the TCAM module will output the matched prefix[2], 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.

B. A Scalable FISE Design

We incorporated the improvements mentioned in Section II-C and IV-B, such that FISE accommodates more rules, and allow more frequent updates. With the improvements, the source index (see Figure 7(a)) only stores the column address. However, the format of destination index changes (see Figure 7(b)): 1) it has an indicator bit, which is set only if the related destination prefix points to a row in TD-table (see Section II-C); 2) it stores the default entry for the related destination prefix (see Section IV-B).



(a) Source index format (b) Destination index format

Fig. 7: Source and destination indexes format

Within the modified structure, the lookup process changes. After obtaining destination and source indexes. FISE checks the indicator bit, if it is unset, then FISE gets the default index directly. Else if none source prefix gets matched, then FISE gets the default index. Else if a source prefix is matched, then FISE accesses the corresponding cell in TD-table. If the cell is empty, then FISE switches back to the default index, else FISE gets the index value of the cell. Using the obtained index value, FISE looks up in the mapping table, and gets the next hop information. Compared with the original lookup process in Figure 3, the additional steps are processed in CPU, so it does not bring additional accesses in TCAM or SRAM.

[2]Many devices support multiple lookups in parallel [13].

VI. Discussion about Scalability

We admit that although SRAM is larger than TCAM, the basic FISE will bring scalability issues in SRAM in extreme cases. Current largest SRAM chip in the market is 144Mb (288Mb SRAM is on the roadmap of major vendors) other memory products such as RLDRAM can provide similar performances (allows 16 bytes reading with random access time of 15 ns) with memory denominations of 576 Mbit/chip. Suppose multiple chips (line-cards of Bit-Engine 12004 supports 4 SRAM chips while most routers can support 12) is used, and 576Mb storage space is available for TD-table. CERNET2 has about 7000 prefixes in its FIB, thus CERNET2 can even achieve full policies (49 millions rules, need 392Mb SRAM) between all destination and source prefixes with FISE. This greatly improves the traditional structures, which accommodates less than 1 million rules. However, this is still not enough for larger enterprises.

The situation can be improved because 1) Using non-homogenous structure can exclude most destination prefixes from destination table, and not all source prefixes need to be diverted; 2) In the real world, different prefixes usually share the same policy, e.g., prefixes belong the the same university in CERNET2 are treated equally. They can be compressed to a coarser granularity; 3) We can enforce restrictions when adding a row or column into the TD-table. Beside, we are making continuous efforts to further compress the TD-table.

VII. Performance Evaluation

A. Evaluation Setup

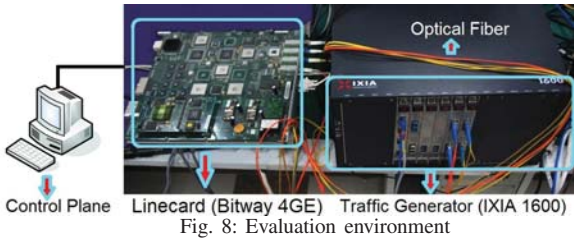


Fig. 8: Evaluation environment

Our evaluation environment is shown in Fig. 8. It has three components: 1) a PC host with a CPU of Intel Core2 Duo T6570 acting as the control plane, 2) a 4GE line-card equipped with both ACL-like and FISE structures, and 3) a traffic generator (IXIA 1600T) with speed of 4Gbps. The traffic generator is connected to the line-card through optical fibers and the line-card 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 speed. The line-card receives the packets, performs lookups and sends the packets back to the traffic generator. Besides, IXIA 1600T can summarize the sending and receiving rate.

We control the forwarding table by the PC host through the serial cable. We update the forwarding table through the pre-defined interfaces. We test update at different frequency, i.e., 500, 5,000, and 50,000 updates per second. The TCAM memory is structured according to L-algorithm [19]. More specifically, prefixes of the same length are clustered together and free space between different clusters is reserved to guarantee fast updates in TCAM.

B. Data Sets

We study two practical scenarios in CERNET2: policy routing and load balancing. Within each scenario, we generate data sets of rules that need to be stored in the forwarding table, and update sequence. Based on real data collected from CERNET2, we can test performance of FISE on IXIA 1600T. Note that problems in these scenarios can be solved by other techniques, e.g., MPLS. However, we focus on forwarding table design in this paper.

1) Scenario 1: Policy Routing in CERNET2: CERNET2 has two international exchange centers connecting to the Internet: IX1 in Beijing and IX2 in Shanghai. During operations, we found that IX1 is very congested with an average throughput of 1.18Gbps (February 2011); and IX2 is much more spared with a maximal throughput of 8.3Mbps at the same time. CERNET2 wants to divert out-going International traffic to IX2.

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: THU (in Beijing, with 38 prefixes), HUST (in Wuhan, with 18 prefixes) and SCUT (in Guangzhou, with 28 prefixes), and forward their traffic to IX2. We thus simulate three FIBs on three routers, Beijing, Wuhan and Guangzhou (we call each FIB PR-BJ, PR-WH, and PR-GZ).

We generate the update sequence on the router of WH 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 II: Data sets overview

2) Scenario 2: Load Balancing in CERNET2: In the future, we need a more dynamic mechanism to balance the load between IX1 and IX2. We collected about 1TB NetFlow traffic data during one month in 2012 from three major routers. In Fig. 9, we show the bandwidth utilizations of both IX1 and IX2 during the month. We can see that IX1 is much more congested than IX2.

We try to redistribute each *macro flow*, identified by a destination and source prefix pair, to different exchange centers, such that load is optimally balanced. The problem is known to be NP-hard, thus we use the greedy first-fit algorithm, which 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 WH (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. We generate the update sequence by computing a new load balancing forwarding table every hour.

C. Evaluation Results

1) Forwarding Table Size: We evaluate the storage space that FISE consumes for all forwarding tables, and the stor-

age space after compression and adopting non-homogeneous structure. As a comparison, we set the ACL-like structure as a benchmarks. We compare FISE and ACL-like in each step. We also compare with SPliT [17] structure, which first lookups in one dimension, outputs a sub-table, and merges sub-tables if they are the same. Because SPliT undergoes totally different steps, we only compare FISE and SPliT in the final step.

In Fig. 12, we show the consumed TCAM and SRAM storage space of each forwarding table.

Basic FISE and ACL-like Structure: In Fig. 12(a), we can see that the TCAM space in FISE can be 1/50 as compared to ACL-like structure. For example, in PR-WH, FISE consumes 1Mb TCAM storage, while ACL-like structure consumes more than 72Mb. In the LB scenarios, FISE 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. 12(e), we can see that, in the PR scenario, the SRAM space in FISE can be 1/40 as compared to ACL-like structure. For example, in PR-WH, FISE consumes 3Mb SRAM storage, while ACL-like structure consumes 125Mb. In the LB scenario, FISE consumes more SRAM storage. This is because in the PR scenario, although many rules exist, the TD-table is very dense, and nexthop index further condenses the nexthop information. In LB scenario, the TD-table is sparser.

Compression: In Fig. 12(b), 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, FISE still consumes much less TCAM storage than ACL-like structure. In Fig. 12(f), we show the SRAM space after compression. We can see that FISE can further compress SRAM after TCAM compression. For example, SRAM storage of PR-WH is compressed to be less than 90K bits. This is because 1) the flexible mapping structure of FISE; 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-Homogenous Structure: In Fig. 12(c), we show the TCAM space with non-homogeneous structure. Non-homogeneous structure does not save TCAM storage in FISE 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 FISE, we can logically divide the table into two parts. In Fig. 12(g), we show the SRAM space with non-homogeneous structure. We can see that, with non-homogeneous structure, FISE consumes much less SRAM than ACL-like structure in all forwarding tables.

Compression with Non-Homogenous Structure: In Fig. 12(d) and 12(h), we apply non-homogeneous structure and compression techniques to FISE and ACL-like structure. The resulting tables are the smallest among all tables. We can see that TCAM and SRAM spaces in FISE are much smaller than ACL-like structure. The improvement in SRAM is quite large compared to non-homogenous structure only, because there still exists redundancies after using non-homogenous structure.

We also compare FISE with SPliT, in Fig. 12(d), we see that in the PR scenario, although SPliT improves ACL-like structure, it still consumes much more TCAM storage than FISE. For example, on PR-WH, SPliT consumes more than 4.5Mb TCAM while FISE only consumes 800Kb. This is because SPliT does not fully eliminate the "multiplication" factor in TCAM while FISE does. In the LB scenario, the improvement is not obvious, because only a few two dimensional rules exist. In Fig. 12(h), we see that in the PR scenario, SPliT consumes much more SRAM storage than FISE. In the LB scenario, SPliT also consumes similar SRAM storage with FISE.

From the above evaluations, we can conclude that compared to other structures, FISE can save large TCAM storage space. Although the SRAM storage space may be larger initially, through various flexible techniques, SRAM storage space of FISE can be largely reduced. Note that the metric of storage can be converted to monetary cost and power consumption, thus ISPs can save money/power [26].

2) Lookup Speed and Update: We show that FISE can achieve line-card speed, and compare its update performances with ACL-like structure[3].

Lookup Speed: In Fig. 10, we show the lookup speed without updates. We can see that without updates, both sending and receiving rates reach line speed (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 line-card we use, and it could be higher with better line-cards.

TCAM Accesses During Update: We evaluate the number of accesses to TCAM because updates in TCAM will interrupt the lookup. In Fig. 13(a), we show the number of TCAM accesses per 100 updates in PR and LB scenarios. We see that the number of TCAM accesses that FISE causes are three orders less than ACL-like structure. For example, in the PR scenario, FISE causes 2-3 TCAM accesses per 100 updates while ACL-like structure causes several thousands. This is because FISE stores much less entries in TCAM.

In Fig. 13(b) and 13(c), we show the lookup speed, i.e., receiving rate on the traffic generator, of FISE with different update frequencies during 5 minutes. In Fig. 13(b), we can see that in the PR scenario, FISE has no influence on lookup while ACL-like structure degrades the lookup speed by 7% in the worst case. This is because FISE causes much fewer accesses to TCAM. In Fig. 13(c), we can see that in the LB scenario, FISE does influence the lookup speed when there are 50,000 updates per second, however, the influence is still much smaller than ACL-like structure.

We conclude that FISE structure will not impose high update burden on lookups. In the PR scenario, all updates can be finished in less than 10 seconds without influencing lookups, 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 1s without influencing lookups.

[3]SPliT does not have an online incremental update algorithm.

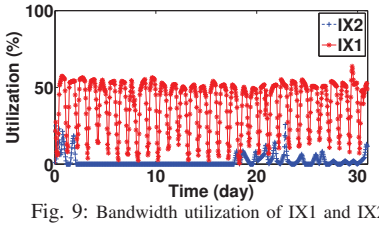


Fig. 9: Bandwidth utilization of IX1 and IX2

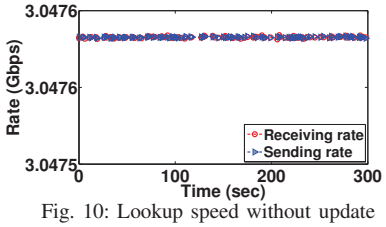


Fig. 10: Lookup speed without update

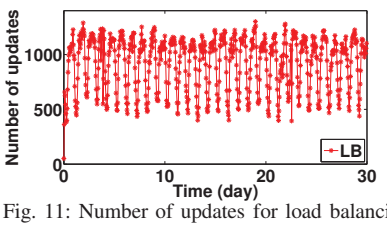
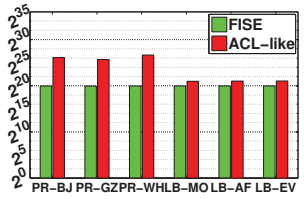
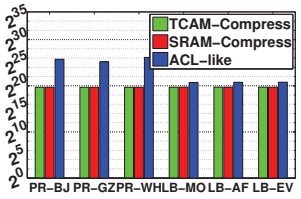


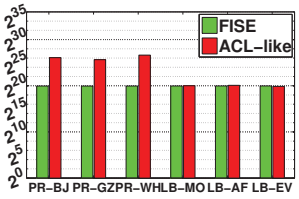
Fig. 11: Number of updates for load balancing



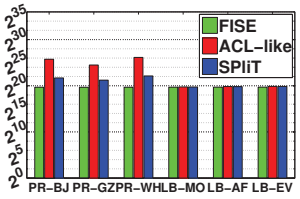
(a) TCAM with basic structure



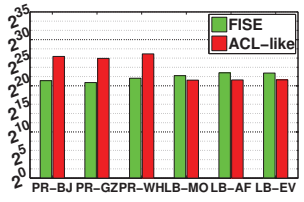
(b) TCAM after compression



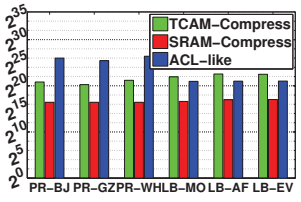
(c) TCAM with non-homogeneous



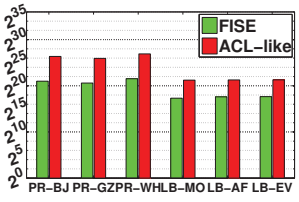
(d) TCAM after compression, with non-homo



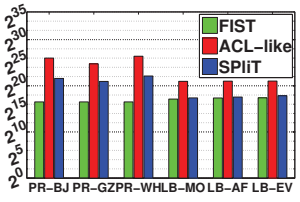
(e) SRAM with basic structure



(f) SRAM after compression

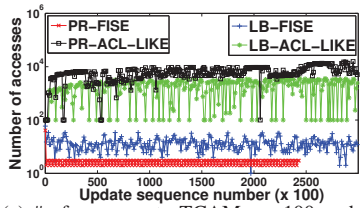


(g) SRAM with non-homogeneous

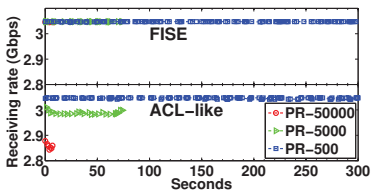


(h) SRAM after compression, with non-homo

Fig. 12: Size of each forwarding table

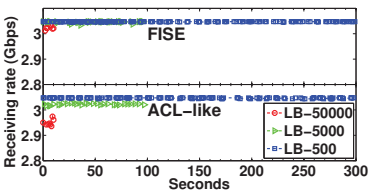


(a) # of accesses to TCAM per 100 updates

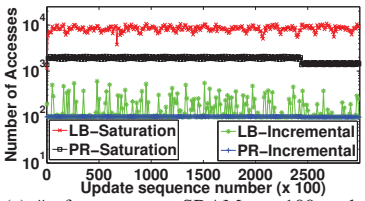


(b) Lookup speed with updates for PR

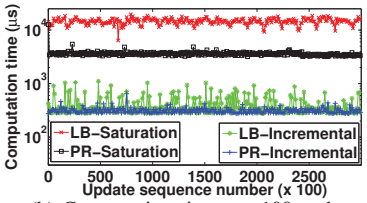
Fig. 13: Lookup speed with updates



(c) Lookup speed with updates for LB



(a) # of accesses to SRAM per 100 updates



(b) Computation time per 100 updates

Fig. 14: Comparison between incremental updates and TD-Saturation()

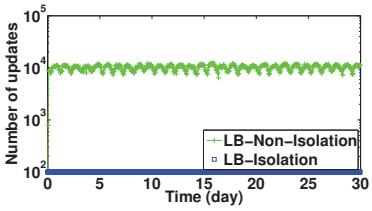


Fig. 15: Isolation VS. non-isolation

SRAM Accesses During Update: In Fig. 14, we show the number of accesses to SRAM within incremental update and TD-Saturation(). We can see that for both PR and LB scenario, incremental update causes much less accesses to SRAM. This is because during each update, TD-Saturation() has to reset all conflicted cells while incremental update only has to reset a small part of them. For example, in the LB scenario, incremental update causes 600 accesses in SRAM, while TD-Saturation causes 10,814. In the PR scenario, incremental update causes only 100 accesses, this is because in the PR scenario, the source table are composed of prefixes from U1 and U2, whose prefixes are disjoint except for two prefixes (240c::/28 and 240c:3::/32). Thus update a cell in TD-table brings almost none conflicted cells.

In Fig. 14b, we also show the computation time per 100 updates for both incremental update and TD-Saturation(). The result is similar with Fig. 14, because more accesses to SRAM indicates more cells that have to be computed.

In Fig. 15, we show the number of accesses to SRAM with and without isolating default entry in source tables. We only consider the LB scenario, because PR scenario is a special case where all nodes in the colored tree of any destination prefix are black, thus isolating default entry has no effect. In the LB scenario, we randomly insert 100 updates on the default next hops of destination prefixes, after each hour when load balancing is carried out. In Fig. 15, we can see that with isolation, each update cause none additional accesses to SRAM, because we only have to update the TCAM and destination index. However, without isolation, each 100 updates cause 10,000 accesses to SRAM, because we also have to update the conflicted cells in TD-table.

VIII. Related Work

With increasing demands for flexible routing policies in enterprise networks, many routing solutions have been proposed

throughout the history, e.g., PBR [5], MTR [8], TCR [14][2] and recent software-defined network (SDN). Most of these solutions focus on re-designing the control plane. Our work re-designs the forwarding plane, and is orthogonal with them.

The simplest extension to support rich policies is using multiple one dimensional forwarding tables [14][8], However, this solution does not scale. Improved solutions can be divided into two broad categories: CAM-based and algorithmic solutions [25]. Here, we focus on CAM-based solutions.

CAM-based, especially TCAM-based solutions are the de facto standard in industry. Most enterprise networks uses ACL-like structure, which is 'fat' in TCAM and 'thin' in SRAM [16]. However, TCAM-based solutions are limited by its capacity [17]. What is worse, TCAM is highly customized, there are limited techniques we can use to compress it. The most popular technique is aggregation [11][15]. In [23], the optimal two dimensional forwarding table compression is studied. In [15], a non-prefix approach that re-orders the ternary strings in prefixes to compress TCAM is studied. Compared with them, we move the storage from TCAM to SRAM, thus 1) TCAM storage is reduced; 2) More compression techniques can be used in SRAM.

There are studies proposing new structures to reduce the "multiplicative" effect in TCAM [16]. In [17], a scheme called SPliT first lookups in a one dimensional table storing destination prefixes, and outputs a sub-table. Thus, it can merge different sub-tables if they are the same. Compared with SPliT, we make one step further and fully eliminate the effect.

More works are proposed for algorithmic solutions, such as trie-based, decision-tree, and bitmap-based approaches [25]. However, they suffer from non-deterministic performance and do not scale well [13]. Although we focus on CAM-based, we borrow ideas from other non-CAM solutions. Bit-vector linear search [9] performs individual lookups in each dimension, each dimension will output a $O(n)$ length vector representing matched rules. By interseting bit-vectors, the algorithm computes the final result. Cross-producting [21] extracts the elements in each dimension, and stores all combinations in a database. Based on their ideas, we formally organize the rules into TCAM and a compact matrix in SRAM, which is simple and provides deterministic lookup speed.

IX. Conclusion

In this paper, we put forward a new forwarding table structure called FISE, where forwarding decisions are based on both destination and source addresses. Our focus is to accommodate the increasing number of policies in enterprise networks, which is also a practical concern of CERNET2. Through separation between TCAM and SRAM, FISE can greatly reduce the TCAM storage and keep fast lookup speed.

We implement the FISE-based forwarding table on the line-card of a commercial router. Our design does not need any new device. We also made comprehensive evaluations with the real design and data sets from CERNET2. The results showed that the performances of FISE are promising.

Here, we focus on layer-3 two dimensional table, due to the importance of source address in routing. It is also an initial step towards higher dimensional forwarding in our future work.

References

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

[2] F. Baker. Routing a traffic class. Internet Draft, Jan 2012. draft-baker-fun-routing-class-00.

[3] T. Benson, A. Akella, and D. A. Maltz. Mining policies from enterprise network configuration. In *Proc. ACM IMC'09*, Chicago, IL, Nov 2009.

[4] M. Boutier. Source-specific routing. Internet Draft, Jul 2013. draft-boutier-homenet-source-specific-routing-00.

[5] Cisco. *Policy-Based Routing (white paper)*, 1996.

[6] 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.

[7] J. Fu and J. Rexford. Efficient ip-address lookup with a shared forwarding table for multiple virtual routers. In *Proc. ACM CoNEXT'08*, Madrid, Spain, Dec 2008.

[8] Juniper. *Multi-topology routing (white paper)*, Aug 2010.

[9] 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.

[10] A. Lindem, S. Mirtorabi, A. Roy, and F. Baker. Ospfv3 lsa extendibility. Internet Draft, May 2013. draft-acee-ospfv3-lsa-extend-01.

[11] 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.

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

[13] 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.

[14] F. Baker. IPv6 Source/Destination Routing using OSPFv3. Internet Draft, Feb 2013. draft-baker-ipv6-ospf-dst-src-routing-00.

[15] 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.

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

[17] 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.

[18] 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.

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

[20] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: network processing as a cloud service. In *Proc. ACM SIGCOMM'12*, Helsinki, Finland, Aug 2012.

[21] 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.

[22] Y. Sung, X. Sun, S. Rao, G. Xie, and D. Maltz. Towards systematic design of enterprise networks. *Networking, IEEE/ACM Transactions on*, 19(3):695 –708, 2011.

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

[24] Z. A. Uzmi, M. Nebel, A. Tariq, S. Jawad, R. Chen, A. Shaikh, J. Wang, and P. Francis. Smalta: practical and near-optimal fib aggregation. In *Proc. ACM CoNEXT'11*, Dec 2011.

[25] G. Varghese. *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices*. Morgan Kaufmann, Waltham, MA, 2005.

[26] S. Yang, M. Xu, D. Wang, and J. Wu. Two dimensional router: Design and implementation. Technical report, Tsinghua University, May 2013. www.wdklife.com/tech.pdf.

[27] M. Yu. *Scalable Management of Enterprise and Data-Center Networks*. PhD thesis, Princeton University, Sep 2011.