FISE: A Forwarding Table Structure for Enterprise Networks

Shu Yang, Laizhong Cui, Xinhao Deng, Qi Li, Yulei Wu, Mingwei Xu, Dan Wang, Jianping Wu

***Abstract*—With increasing demands for more flexible services, the routing policies in enterprise networks become much richer. This has placed a heavy burden to the current router forwarding plane in support of the increasing number of policies, primarily due to the limited capacity in TCAM, which further hinders the development of new network services and applications. The scalable forwarding table structures for enterprise networks have therefore attracted numerous attentions from both academia and industry.**

To tackle this challenge, 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 and flexible SRAM. A set of schemes are progressively designed, to compress storage of forwarding rules, and maintain correctness and achieve line-card speeds of packet forwarding. We further design an incremental update algorithm that allows less access to memory. The proposed scheme is validated and evaluated through a realistic implementation on a commercial router using real datasets. Our proposal can be easily implemented in the existing devices. The evaluation results show that the performance of forwarding tables under the proposed scheme is promising.

***Index Terms*—Enterprise Network, Forwarding Table Structure, Routing Policy**

I. Introduction

Enterprise networks are facing great challenges due to the ever-increasing demands for much more flexible routing policies in enterprises [2]. A typical enterprise network can host thousands of routers [3], and its management is responsible for 80% of the IT budget and 62% of the outages [4]. Different from backbone networks, with the main purpose of providing reachability services, enterprise networks need to support much more fine-grained routing policies. For example, 1) **Multi-homing:** an enterprise should be able to deliver the traffic of IPTV users to one upstream ISP, and the traffic of VoIP users to another upstream ISP; 2) **Access control:** a hospital needs to implement strict access control to gurantee the security of patients' electronic records [5]; 3) **Performance:** a bank should choose secure paths for its bank applications, and low-latency paths for its financial applications [6]. Other examples in enerprise networks include load balancing, network virtualization, etc.

There exist many solutions to support flexible policies for traffic control. For example, policy-based routing (PBR) [7] implements 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) [9, 10] that adds more information, e.g., the 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 accommodate the increasing number of routing policies. Nevertheless, current solutions are not scalable. For example, MTR uses a separate forwarding table for each topology, and it can only support a limited number, i.e., 32 in most cases [8], of topologies while current enterprise networks require much more [11]; TCR recommends using one forwarding table per source prefix, which scales even worse than MTR and is only suitable for small networks.

Many other solutions, like PBR using the traditional Cisco Access Control List (ACL) structure (named ACL-like structure thereafter) in TCAM. A typical forwarding table with ACL-like structure is shown in Table I. For the purpose of illustration, we use 4-bit IP addresses. The destination and source prefixes are concatenated as an entry in TCAM. When a packet with the destination address of 1011 and the source address of 1111 arrives, the router 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 prefixes.

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

This paper is an extendable version of the INFOCOM 2014 paper [1]

Shu Yang and Laizhong Cui are with the College of Computer Science & Software Engineering, Shenzhen University, Shenzhen, PR.China and with the Guangdong Laboratory of Artificial-Intelligence and Cyber-Economics (SZ), Shenzhen University.

Laizhong Cui is the corresponding author(Email: cuilz@szu.edu.cn).

Xinhao Deng, Qi Li, Mingwei Xu and Jianping Wu are with the department of Computer Science and Technology, Tsinghua University.

Yulei Wu is with the College of Engineering, Mathematics and Physical Sciences, University of Exeter.

Dan Wang is with the department of computing, The Hong Kong Polytechnic University.

This work has been partially supported by National Key R&D Program of China under Grant No.2018YFB1800302, National Natural Science Foundation of China under Grant No.61772345, No.61625203, No.61902258 and No.61832013, and Tencent "Rhinoceros Birds" - Scientific Research Foundation for Young Teachers of Shenzhen University.

prefixes and 100 source prefixes. This results in a requirement of at least 600,000 entries in TCAM, and this number is also likely to increase in the future. Even after being compressed, the table size is still too large for CERNET2. Many modern enterprise networks face similar problems after widely deploying lots of security, QoS and privacy functions in networks [11]. It is well 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 [12]. Thousands of rules in ACL will bring large overheads to an enterprise [13]. As a matter of fact, the largest TCAM chip in market can only accommodate 1 million IPv4 prefixes [14]. However, existing forwarding plane solutions do not scale well in TCAM, and this has become a bottleneck for developing new services in enterprises [14, 15].

TABLE I: A Two Dimensional Forwarding Table Example

#	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

In this paper, we design a new forwarding table structure, which can handle the expanded policies in enterprise networks. The proposed new forwarding table structure is called FISE (**FI**B **S**tructure for **E**nterprise). The key of FISE is a separation of TCAM and SRAM. SRAM has larger memory, and it is 10 times cheaper and consumes 100 times less energy than TCAM. We move the storage from TCAM to flexible, cheap and power-saving SRAM. An example of FISE is shown in Fig. 1. FISE stores destination and source prefixes in two separate TCAM tables, and keeps 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.

In enterprise networks, it is important to enforce fine-grained routing policies to achieve diversified performance goal. Thus, this paper aims to develop a sophisticated design to implement such scalable policy enforcement with less overhead. By leveraging both TCAM and SRAM, we can achieve fast rule lookup while significantly reducing the memory for the next-hop storage. In particular, it can substantially reduce the number of TCAM entries to $N + M$ while flexibly enforcing various policies without rule redundancies.

However, there exist several challenges in the development of FISE: 1) we need to guarantee the line-card speeds and correctness of packet forwarding, and it is difficult; 2) because, in principle, an update in FISE may incur multiple accesses in memory, we need to develop incremental update algorithms to minimize such accesses; and 3) in practice, we want to make FISE more scalable and

TABLE II: Major notations used in the paper.

Notation	Meaning
Γ	Two dimensional table that stores the indexed nexthop of each rule
$\Gamma(x, y)$	The cell in x^{th} row and y^{th} column of table Γ
Λ	A New two dimensional table obtained after updating the table Γ
$\Lambda(x, y)$	The cell in x^{th} row and y^{th} column of table Λ
s	The source address
d	The destination address
p_s	The source prefixes for s
p_d	The destination prefixes for d
a	The action of a rule
P_s	The set of source prefix
P_d	The set of destination prefix
$\mathcal{R}$	The set of forwarding rules to be stored

accommodate a number of rules in the order of millions, which is 100 times of that of today. In this paper, we present a comprehensive study on FISE and progressively tackle these problems. The FISE is implemented on a commercial router, i.e., Bit-Engine 12004, and a set of practical implementation designs are elaborated. It is worth noting that through careful redesign of the hardware logic, FISE can be easily implemented on the existing devices, instead of requiring new devices. Comprehensive evaluations with the real implementation are carried out, using the real topology, FIB, prefix and traffic data from CERNET2. The results show that FISE can achieve line-card speeds of packet forwarding, save TCAM and SRAM storage, and bring acceptable update burden.

The rest of the paper is organized as follows: Section II is devoted to FISE design. The FISE structure is introduced, and its correctness for the forwarding operations is proved. The TCAM and SRAM compression schemes are presented in Section III. Section IV is devoted to the design of FISE incremental updates. The implementation design is elaborated in Section V, which can improve the adoptation of FISE in practical situations. Section VI shows the implementation of FISE on a commercial router. The evaluation of FISE is carried out in Section VII. The related work is presented in Section VIII. Finally, we conclude this paper in Section IX.

II. FISE Structure and Lookup

A. *The Matching Rule*

A conventional forwarding table means the table uses a traditional ACL structure to organize rules, e.g., in Cisco and Juniper routers, while a conventional router indicates the router having the structure of a conventional forwarding table. In conventional router, the LMF rule is used to decide which destination prefix is matched. Here, we first present the definition of the matching rule. Let d and s denote the destination and source addresses, and 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. Each entry of the storage is a 3-tuple (p_d, p_s, a). In Table II we summarized the major notations in the paper.

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 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 the next hop a.*

We match destination prefixes first, rather than match destination and source prefixes with the same priority [16], because we have to guarantee reachability and avoid routing loops [17][18]. Note that the structure in our FISE design is symmetric. Thereby we can define the priority of prefix based on the order of rules according to application scenarios.[1].

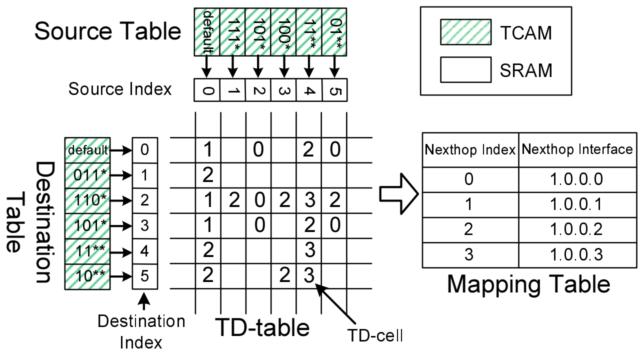


Fig. 1: FISE: A forwarding table structure for enterprise networks

B. FISE Basics and Correctness

1) FISE Basics: The key idea is a separation of TCAM and SRAM (see Fig. 1). The destination and source 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.

FISE has two tables in TCAM and another two tables in SRAM. In TCAM, one table stores the destination prefixes mapping to a *destination index* (we call it *destination table* thereafter), and another table stores the source prefixes mapping to a *source index* (we call it *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 correspond to a TD-cell in SRAM. The other table in SRAM stores the mapping relations of index values and the 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. From this index value, a is stored in the corresponding position of the mapping table.

An example is shown in Fig. 1. For (110*,11**, 1.0.0.3), 110* is stored in the destination table and points to the destination index 2, which is associated with the 3nd row; and 11** is stored in the source table and points to the source index 4, which is associated with the 5th column. We can obtain that, in the TD-table, the cell (110*, 11**) in the 3nd row and the 5th column has the index value of three. In the mapping table, the next hop with the index value 3 is 1.0.0.3.

[1]Source prefix match first rule is also used in some special designs, e.g., egress-routing in multi-homing [9].

Theorem 1. *The TCAM storage space of FISE is $O(N + M)$ bits. The SRAM storage space of FISE is $O(N \times M \times log(P))$ bits, where P is the size of the mapping table.*

See proof in [19]. Clearly, FISE migrates the "multiplication" factor into SRAM, rather than eliminate it. Such migration is based on the following observations: 1) TCAM storage capacity is much smaller than SRAM; 2) TCAM is 10-100 times more expensive than SRAM; 3) and TCAM consumes 100+ times more power than SRAM [20, 21, 22].

This migration does not slow down the forwarding speed. The dominant factor for forwarding speed is TCAM lookup, which can be maintained by FISE. In FISE, there are additional SRAM accesses. With pipelining scheme, which is commonly used in routers, the lookup speed is the same as the one of conventional routers. The delay of each lookup is longer, but it is only a few nanoseconds and thus it is acceptable.

Moreover, SRAM is more flexible than TCAM, such that we can develop various compression algorithms in SRAM to further reduce TD-table storage.

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

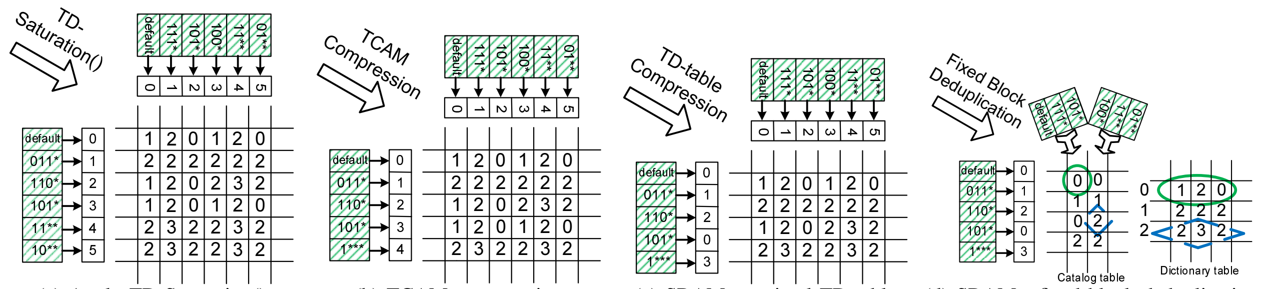
Note that after insertion, there will be empty cells. Consider a packet with destination address 1011 and source address 1111 arrives at the router, according to definition 1, the destination prefix 101* will first be matched. There are four rules (including the default rule) associated with the destination prefix 101*. Source prefix 11** will then be matched. This leads to the 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, the cell $(101^*, 111^*)$ (4^{th} row and 2^{nd} column) in TD-Table does not have any index value. Intrinsically, for prefix pairs (p_d, p_s), if the length of a source prefix p'_s that is longer than p_s, when a packet that matches both rules arrives, the cell (p_d, p'_s) with the longer prefix rather than (p_d, p_s) will be matched.

Definition 2. ***Conflicted cell****: For a TD-cell (p_d, p'_s), if the value of the cell is null but a rule associated with (p_d, p_s, a) (where p_s is a sub-prefix of p'_s) exists, (p_d, p'_s) is a conflicting cell.*

To address the problem, we develop an algorithm TD-Saturation() to saturate the conflicted cells with an appropriate index value. As an example, using this algorithm, the TD-table of Fig. 1 becomes Fig. 2(a). The time complexity of Algorithm 1 is $O(N \times M)$, and the space complexity of it is $O(N \times M)$. We can see that, routers preform TD-cell saturation to pre-compute the next-hops for each (destination, source) prefix pair, which provides possibility for line-card speed lookup with the FISE structure.

Theorem 2. *FISE (with* TD-Saturation*()) correctly handles the rule defined in Definition1.*

See proof in [19].



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

Fig. 2: FISE forwarding table storage

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

```
1  begin
       // $\mathcal{R}$ is the set of forwarding rules to be stored
2      foreach $p_d, p_s$ do
3          if $\nexists(p_d, p_s, a) \in \mathcal{R}$ then
4              $\mathcal{S} = \{(\bar{p_s}, \bar{p_d}, \bar{a}) \in \mathcal{R} | \bar{p_d} = p_d\}$
5              $\mathcal{S}' = \{(\tilde{p_s}, \tilde{p_d}, \tilde{a}) \in \mathcal{S} | \tilde{p_s}$ is a prefix of $p_s\}$
6              Find $(p''_s, p''_d, a'') \in \mathcal{S}', \forall(p'_d, p'_s, a') \in \mathcal{S}', p'_s$ is a
               prefix of $p''_s$
7              Fill the cell $(p_d, p_s)$ with index value of $a''$.
```

C. A Non-Homogeneous FISE 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. 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 by using an indicator bit in the destination index to separate them. More details are presented in Section VI.

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 performance as the conventional routers for each lookup operation.

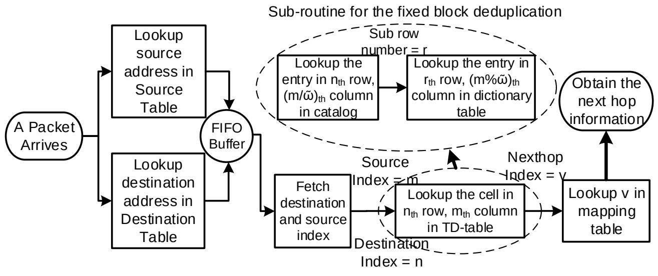

Sub-routine for the fixed block deduplication
Sub row number = r
Lookup source address in Source Table
Lookup destination address in Destination Table
A Packet Arrives
FIFO Buffer
Lookup the entry in n_{th} row, $(m/\bar{\omega})_{th}$ column in catalog
Lookup the entry in r_{th} row, $(m\%\bar{\omega})_{th}$ column in dictionary table
Obtain the next hop information
Fetch destination and source index
Source Index = m
Destination Index = n
Lookup the cell in n_{th} row, m_{th} column in TD-table
Nexthop Index = v
Lookup v in mapping table

Fig. 3: Lookup process in FISE

The lookup action is shown in Fig. 3. When a packet arrives, the router matches the destination and source prefixes in parallel in the destination and source tables in TCAM. The destination table and source table then each outputs the SRAM addresses that point to the destination index and source index, respectively. 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 router looks up the mapping table, and returns the nexthop.

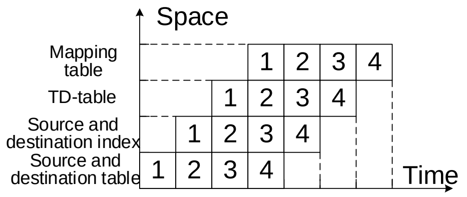


Fig. 4: Space-time diagram of the lookup process pipeline

Observation 1. *The lookup speed of FISE is one TCAM plus three SRAM clock cycles.*

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 pipelining each individual lookup operation.

Fig. 4 shows the process of pipeline processing for four packets numbered with 1, 2, 3, and 4. When a packet arrives, the router looks up the source and destination addresses in the source and destination tables, respectively, and obtains the source and destination indexes to look up the corresponding cell in the TD-table. It finally obtains the next hop in the mapping table according to the cell value.

This pipeline is possible since we have a saturated TD-table. Pipelining 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 2. *The lookup speed of the FISE routers (with pipelining) is the same as conventional forwarding tables.*

In practice, the SRAM clock cycle is smaller than TCAM cycle [23].[2] Also, the bottleneck of a router is normally during delivering packets through the FIFO. These facts further validate that FISE lookup speed is comparable to conventional routers.

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

III. Forwarding Table Compression

The FISE 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 the equivalent class of two tables. As such, we can select another table in the equivalent class that has the minimum size.

Let P_d and P_s be the set of destination prefix and source prefix, 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 a source index). Let $\Gamma(x, y)$ denote the cell in x^{th} row and y^{th} column of TD-table. We use a 5-tuple $\{P_d, P_s, f_r(\cdot), f_c(\cdot), \Gamma(\cdot,\cdot)\}$ to denote a FISE table.

Definition 3. *Two FISE tables are* ***equivalent*** *if any packet will be forwarded to the same nexthops indexed by two FISE tables according to the matching rules in Definition 1.*

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

Algorithm 2: CompTCAM($P_d, P_s, f_r(\cdot), f_c(\cdot), \Gamma(\cdot,\cdot)$)

```
    Output : $\{P'_d, P'_s, f'_r(\cdot), f'_c(\cdot), \Lambda(\cdot,\cdot)\}$
1   begin
2       Eliminate prefixes that will never be matched
3       $\forall p_d \in P_d, \mathcal{DF}(p_d) = SHA\text{-}1(\overrightarrow{\Gamma(f_r(p_d), \cdot)})$
4       $\{P'_d, \mathcal{DF}'(\cdot)\} \leftarrow ORTC(P_d, \mathcal{DF}(\cdot))$
5       $\forall p'_d \in P'_d, f'(p'_d) \leftarrow f(p_d), \exists p_d \in P_d, \mathcal{DF}'(p'_d) = \mathcal{DF}(p_d)$
6       $\forall p_s \in P_s, \mathcal{SF}(p_s) = SHA\text{-}1(\overrightarrow{\Gamma(\cdot, f_c(p_s))})$
7       $\{P'_s, \mathcal{SF}'(\cdot)\} \leftarrow ORTC(P_s, \mathcal{SF}(\cdot))$
8       $\forall p'_s \in P'_s, f'(p'_s) \leftarrow f(p_s), \exists p_s \in P_s, \mathcal{SF}'(p'_s) = \mathcal{SF}(p_s)$
9       $\forall p'_d \in P'_d, p'_s \in P'_s, \Lambda(f'_r(p'_d), f'_c(p'_s)) \leftarrow \Gamma(f'_r(p'_d), f'_c(p'_s))$
```

1) TCAM Compression: For TCAM compression, there exists a dynamic programming based algorithm Optimal Routing Table Constructor (ORTC) [24], which computes the minimized TCAM for a set of prefixes in one dimensional router. We separately merge the entries in the source and destination table with ORTC to compress TCAM.

We develop algorithm CompTCAM(), based on the algorithm Optimal Routing Table Constructor (ORTC) [24] which computes the minimized TCAM for a one dimensional forwarding table. In Algorithm 2, we first map each row/column vector to a scalar using SHA-1 function. We then apply ORTC separately to destination and source tables in TCAM, and compute the minimized TCAM for the corresponding rows and columns. Finally, we replace the original TCAM with the compressed TCAM.

Theorem 3. *CompTCAM(), applying ORTC twice on FISE, leads to the minimum total TCAM storage. The complexity of CompTCAM() is $O(N \times M)$*

See proof in [19]. For example, after TCAM compression, the forwarding table in Fig. 2(a) becomes Fig. 2(b).

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 a FISE table $\{P_d, P_s, f_r(\cdot), f_c(\cdot), \Gamma(\cdot,\cdot)\}$, we can find an equivalent forwarding table $\{P'_d, P'_s, f'_r(\cdot), f'_c(\cdot), \Lambda(\cdot,\cdot)\}$ to compress the table such that the storage space, i.e., $N \times M$ (where N is the number of rows and M is the number of columns in the TD-table), is minimized.*

In this stage, we define the optimal TD-table compression is the TD-table without duplicated rows (columns). To find the optimal TD-table compression, we 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 unnecessary in FISE structure since we can merge rows (columns) as long as we make their destination (source) indexes the same.

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

Theorem 4. *Eliminating the duplicated rows and columns computes the optimal TD-table compression.*

See proof in [19]. After eliminating duplicated rows (columns), the table in Fig. 2(b) becomes Fig. 2(c).

Although we can find the optimal TD-table compression, the 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 [25].

Problem 2. ***Optimal SRAM Compression:*** *Given a TD-table, find a minimized representation of it such that 1) given an x^{th} row and y^{th} column, it outputs the same nexthop index with TD-table; 2) we can find a TD-cell in constant time.*

This problem is NP-complete (see proof in [19]). Due to the complexity of the problem, we solve it with a heuristic approach. More details are illustrated in Section V-A.

IV. FISE Incremental Update

Although TD-Saturation() guarantees the correctness of FISE. It needs to re-compute all conflicted cells, and re-write them in SRAM once an update happens. Suppose that there are 10,000 source prefixes, and 500 updates are performed 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 $\Gamma(\cdot,\cdot)$ to denote the TD-table. Let P_d and P_s be the set of destination prefix and source prefix, 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 a source index). Let $\Gamma(x, y)$ denote the cell in x^{th} row and y^{th} column of TD-table. The Action(p_d, p_s, a) represents the change of the value of cell (p_d, p_s) in the TD-table. This operation is performed when new data is added into the cell (p_d, p_s), e.g., the value is updated or the cell is removed.

Problem 3. ***Optimal transformation:*** *Given a TD-table $\Gamma(\cdot,\cdot)$ and* Action*(p_d, p_s, a), find a new TD-table $\Lambda(\cdot,\cdot)$ which is achieved by updating the TD-table with* Action*(p_d, p_s, a), such that $|\{(p_d, p_s)|\Gamma(f_r(p_d), f_c(p_s)) \neq \Lambda(f_r(p_d), f_c(p_s))\}|$ 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 a color tree data structure to organize the cells. With this color tree, we 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 the conventional router also stores data structure such as to organize prefixes. The storage in DRAM is much larger and cheaper, so the extra burden caused by our algorithms is acceptable. We will prove that our algorithms can minimize the computation cost and the number of cell rewrites.

A Color Tree Structure and Update Algorithm

In FISE, we enable matching destination prefix at first. In order to avoid matching an empty cell, we need to update all empty cells indicated by p_s that are generated by updating the value of cell (p_d, p_s). Therefore, 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. 5.

Let $\mathcal{B}(p_d) = \{Node(p_s)|\exists(p_d, p_s, a) \in \mathcal{R}\}$ be the set of black nodes, and $\mathcal{W}(p_d) = \{Node(p_s)|\neg\exists(p_d, p_s, a) \in \mathcal{R}\}$ denote the set of white nodes. For example, in Fig. 5, 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 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 $Node(p'_s) \in \mathcal{N}$ meets: 1) $Node(p'_s)$ is a child of $Node(p_s)$; 2) $\nexists Node(p''_s) \in \mathcal{B}(p_d)$, where $Node(p''_s)$ is an ancestor of $Node(p'_s)$ and a child of $Node(p_s)$.*

For example, in Fig. 5, $Node(11**)$ is an ancestor of $Node(111*)$ and a child of $Node(****)$. Intuitively, the domain of a black node is the largest sub-tree that roots at itself and does not contain any other black nodes. In other words, it contains the immediate white descendants and possibly white descendants' white descendants. For example, in Fig. 5, the domain of $Node(****)$ is $\mathcal{D}(101*, ****) = \{Node(****), Node(100*)\}$.

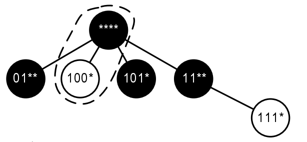


Fig. 5: Color tree $CT(101*)$ for Fig. 1

Theorem 5. *The cell set $\{(p_d, p'_s)|Node(p'_s) \in \mathcal{D}(p_d, p_s)\}$ is the minimum set that is changed by* Action*(p_d, p_s, a).*

See proof in [19]. From Theorem 5, we know that it is enough to find the domain that needs to be updated. We develop update algorithms using depth-first search on the domain. Algorithm Action(p_d, p_s, a) shows the updating process within FISE structure.

Theorem 6. *The complexity of Algorithm 3 is $O(M)$, where M is the number of source prefixes (i.e., the number of color tree nodes).*

Proof: In order to update the cell set $\{(p_d, p'_s)|Node(p'_s) \in \mathcal{D}(p_d, p_s)\}$ that is changed by Action(p_d, p_s, a), all related nodes will be pushed in $nodeStack$ once. Since the number of nodes that are pushed to the $nodestack$ is M in the worst case, the complexity of this step is $O(M)$. The complexity of the update operation on the pop node (i.e., line 11 in Algorithm 3) is $O(1)$. Therefore, we can obtain that the complexity of Algorithm 3 is $O(M)$. ■

Algorithm 3: Action(p_d, p_s, a)

```
1  begin
2      $nodeStack \leftarrow \emptyset$, find $Node(p_s)$ in $CT(p_d)$
3      if Action is Insertion or Update then $value \leftarrow a$
4
5      else if Action is Deletion then
6          $Node(p'_s) \leftarrow$ parent of $Node(p_s)$,
           $value \leftarrow \Gamma(f_r(p_d), f_c(p'_s))$
7
8      Push $Node(p_s)$ into $nodeStack$
9      while $nodeStack \neq \emptyset$ do
10         Pop $Node(\hat{p_s})$ from $nodeStack$
11         $\Gamma(f_r(p_d), f_c(\hat{p_s})) \leftarrow value$
12         for all $childNode \in \mathcal{W}(p_d)$ of $Node(\hat{p_s})$ do
13             Push $childNode$ into $nodeStack$
```

Moreover, in the worse case, a domain contains a row in TD-table, and the complexity of updating table is $O(M)$. 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 [26] [3], TD-table update also does not need to interrupt the lookup process.

V. Practical Considerations

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

A. Further SRAM Compression

*1) **Fixed Block Deduplication**:* We first develop a heuristic approach to further compress SRAM storage. The basic idea is to cut a full row into sub rows and then merge them. Thus we can eliminate more duplicated sub rows.

Merging sub rows requires some modifications on TD-table. Fortunately, SRAM is much more flexible than TCAM to incorporate changes, and one more level of indirection in the TD-table can solve the problem.

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

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 TD-table. We separate the TD-table into a *catalog table* and a *dictionary table* (see an example in Fig. 2(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-to-one mapping.

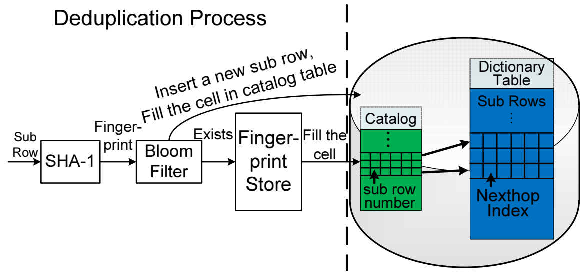


Fig. 6: Storage structure and deduplication process for fixed block deduplication

To construct the new catalog-dictionary tables and achieve higher compression ratio, we use a deduplication procedure that is widely used in data deduplication [27]. A formal flow chart is in [28]. 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 a Bloom filter [29], we can judge whether the sub-row is a duplicated one. If 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 cannot find it in fingerprint store (due to the false positive probability of Bloom filter) or Bloom filter confirms 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. The time complexity of doing fixed block deduplication is $O((M/\tilde{w})N)$. We explain how the Bloom filter and the fingerprint store can be implemented in practice in Section V-A3.

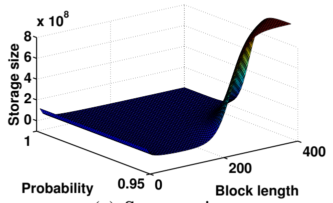
After applying the fixed block deduplication, the TD-table is separated into a catalog table and a dictionary table, where we replace a sub-row with an index in the catalog table, and the index points to a row in the dictionary table. Within an extra indirection memory access, the lookup speed increases by one SRAM clock cycle.

The update algorithm in Section IV is designed for storage without deduplication. The deduplication of the SRAM space, each time update happens, will take much CPU time, although we can tolerate a little more storage space and carry out deduplication at a certain interval, e.g., one hour.

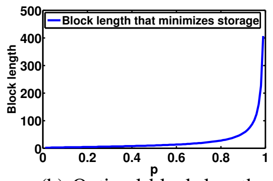
Our design can effectively reduce redundancy in rules. For example, it can significantly reduce the number of the ASes path from one source prefix to a set of destination prefixes [30]. But rules with little redundancy benefit much less, e.g., security rules should be enforced in a fine-grained manner [31].

*2) **Theoretical Analysis**:* Let p_r (or p_c) be the probability that two cells in the same row (or column) are identical. Let p_u denote 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 [28]. 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, Fig. 7(a) shows 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; it then increases, because of the inflation of dictionary table. In Fig. 7(b), we show the block length that minimizes the storage size. The figure presents 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

Fig. 7: Relation between storage size and block length, p

*3) **Parameter Selection**:* In our fixed block deduplication procedure design, two parameters need to be settled during implementation. Let us discuss them in detail.

Bloom filter: The basic Bloom filter is used to accelerate the full storage deduplication process. To avoid unnecessary lookups in the fingerprint store, there exists a summary vector [32], 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. The Bloom filter is widely applied in many areas [27]. We makes the false positive negligible by setting appropriate parameters. To achieve this, we computed the probabilities of false positives in Bloom filter with different settings through a specific method [33]. We found that the false positive will be below 0.2% when m is equal to 512K and k is equal to 4, and it will be below 0.001% if m is equal to 4M and k is equal to 4. Note that, even if a false positive occurs, it will only result in a slight storage overhead. For example, when m is equal to 512K and k is equal to 4, false positive will only bring an additional 0.2% overhead. Therefore, the impact of false positive is acceptable.

Fingerprint store: The fingerprint store organizes all detected fingerprints. To achieve fast searching, it is implemented 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, and the size of a cell in dictionary table to be 8 bits, which is also the size of a TD-cell.

The SHA-1 function of OpenSSL crypto library [34] is used to generate the fingerprints. It can process over 2.5Gb SRAM per second when a sub-row has 500 cells. It can even be accelerated by more than 6 times if implemented in hardware [35]. That is because deduplication overheads are mostly due to the computation of fingerprints [36], so 1Gb SRAM can be deduplicated within about 1 second.

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., Action(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, as it represents the connectivity of destination prefixes. Nowadays, the update frequency on connectivity information can reach tens of thousands per second [26].

We propose to isolate default entries from the source table. We remove these entries from the source table, and instead of being matched explicitly when the full wildcard is hit, the default entry is matched when none entry in the source table is matched. Fig. 8 shows the transformation. More details are illustrated in Section VI.

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 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, it is forwarded to the nexthop of the default entry.

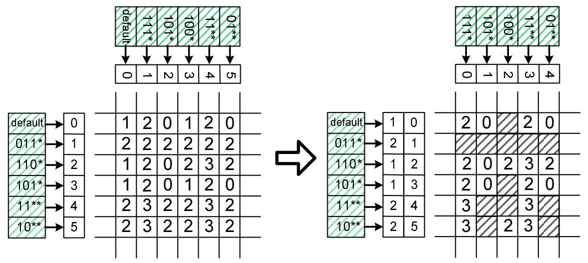


Fig. 8: Isolate default entry from source table

VI. Implementation

FISE is implemented on a commercial router, i.e., Bit-Engine 12004, which supports 4 line-cards. Each line-card has 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. In order to achieve this, we re-design the original destination-based router through rewriting about 1500 lines of VHDL codes with some additional C codes.

A. Router Framework

The logical implementation is depicted in Fig. 9, the packet first arrives at the Interface module. After matching, the TCAM module outputs the matched prefix, and through the TCAM associated SRAM, FPGA obtains 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. FPGA then gets the next hop information and delivers the packet to the next processing module - switch co-process module, which switches the packet to the right interface.

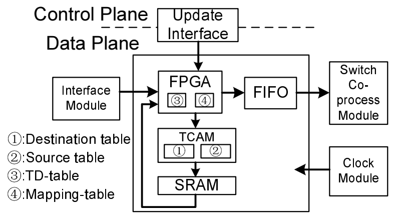
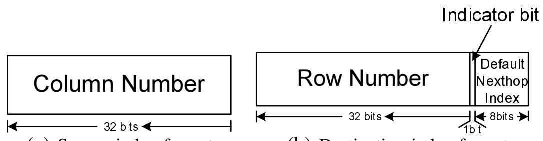


Fig. 9: The framework of router design

B. A Scalable FISE Design

We incorporate the improvements mentioned in Section II-C and V-B, such that FISE accommodates more rules and allows more frequent updates. With this improvement, the source index (see Figure 10(a)) only stores the column address. However, the format of destination index changes (see Figure 10(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 V-B).



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

Fig. 10: 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, FISE gets the default index directly; if none source prefix gets matched, FISE uses the default index; if a source prefix is matched, FISE accesses the corresponding cell in TD-table. If the cell is empty, FISE switches back to the default index, otherwise 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.

VII. Evaluation and Simulation

FISE is evaluated through experiments using real datasets. We also conduct simulations to assist the evaluation of SRAM compression under various settings. In the worst case, the complexity of the FISE scheme is $O(NM)$.

However, due to the real network operation strategies, e.g., default routing is widely configured in networks [9], the actual complexity is far less than $O(NM)$. According to the real dataset, we cannot generate this situation either.

A. Evaluation Environment

Our evaluation environment are composed of three components: 1) a PC host with a CPU of Intel Core2 Duo T6570 acting as the control plane, 2) a 4 Gigabit Ethernet linecard equipped with both ACL-like and FISE structures, and 3) a traffic generator (IXIA 1600T) with speed of 4Gbps. We use an X86 platform PC to act as the main control board of a router to compute and distribute forwarding rules. The architectures are similar to the current mainstreaming router vendors, e.g., Juniper, which also used X86 platforms to implement their main control boards of routers. 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 control the forwarding table by the PC host through the serial cable, update the forwarding table through the pre-defined interfaces and 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 [37]. More specifically, prefixes of the same length are clustered together, and free space between different clusters is reserved to guarantee fast updates in TCAM.

The storage footprint for TCAM and SRAM, and the lookup and update processing speed are also evaluated.

B. Datasets

Two practical scenarios are investigated: policy routing and load balancing. Both scenarios are based on the true demands of CERNET2 [38], which provides the access services for universities/institutions in more than 22 major cities in China. 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 provides more room for the development of policy routing, which may require much more than 1 million rules in the future. However, this is still not enough for larger enterprises. Actually, in our datasets, the number of rules is less than 1 million, e.g., there are less 1,000 source prefixes and less 1,000 destination prefixes. In particular, the reason that we consider a network with 1 million rules is that more rules will be enforced to achieve flexible policy routing in a network deployed with FISE.

Within each scenario, we generate data sets of rules that need to be stored in the forwarding table, and update sequence. Based on the real data collected from CERNET2, we can test the 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 the forwarding table design in this paper.

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 find that CNGI-6IX is very congested with an average throughput of 1.18Gbps (February 2011), and CNGI-SHIX is much more free with a maximal throughput of 8.3Mbps at the same time. CERNET2 thus 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 6407 are foreign prefixes. At the initial stage, we select three universities: Tsinghua University (in Beijing, with 38 prefixes), HUST (in Wuhan, with 18 prefixes) and SCUT (in Guangzhou, with 28 prefixes), and 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, respectively).

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. We show the number of rules in each forwarding table, and the number of updates in each update sequence in Table III.

TABLE III: An overview of the datasets

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

2) Scenario 2: Load Balancing in CERNET2: Fig. 11 (Y-axis is anonymized) shows the bandwidth utilization of CNGI-6IX and CNGI-SHIX. The traffic is very dynamic, thus we need dynamic load balancing mechanisms in the future. As a case study, we collect one Tera-Byte of NetFlow traffic data during Jan, 2012 on routers of Beijing, Shanghai and Wuhan. We will redistribute the flows.

We try to redistribute each *macro flow*, identified by a destination and source prefix pair, to different exchange centers, such that load is balanced. The problem can be reduced to Multi-Processor Scheduling problem [39] 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, respectively). Among them, LB-EV is the largest one, because more traffic should be moved at 22:00 which is the peak traffic point during a day.

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. 12 shows the number of updates per hour in this scenario.

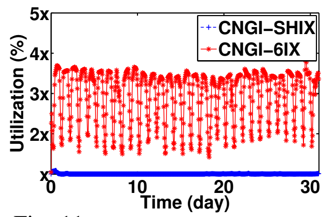


Fig. 11: Utilization of CNGI-6IX and CNGI-SHIX

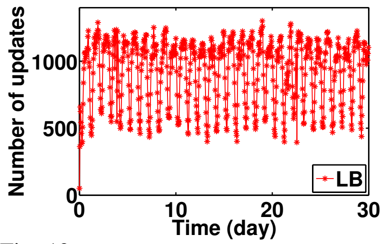


Fig. 12: Number of updates for load balancing

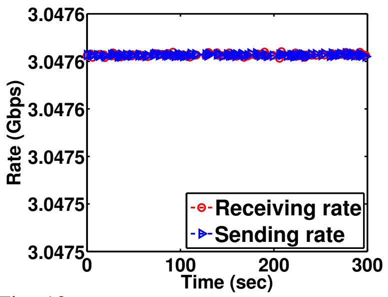


Fig. 13: Receiving/Sending rate without update

Table III summarizes the number of rules and updates (PR: policy routing LB: load balancing). For comparison, we use the ACL-like structure as a benchmark. To update rules, FISE controls outgoing flows by inserting rules in PR scenarios, and periodically changes flow forwarding rules by inserting and removing rules in LB scenarios. Our dataset includes an average of 832 update operations per hour, a maximum of 1301 update operations, and a minimum of 547 update operations per hour.

C. Evaluation Results

1) Forwarding Table Size: We evaluate the storage space that FISE consumes for all forwarding tables, and the storage space after compression and adopting non-homogeneous structure. As a comparison, we set the ACL-like structure as a benchmark. We compare FISE and ACL-like in each step. We also compare FISE with the SPliT [14] 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. 14, we show the consumed TCAM and SRAM storage space of each forwarding table.

Basic FISE and ACL-like Structure: In Fig. 14(a), we can see that the TCAM space in FISE can be 1/50 as compared to the ACL-like structure. For example, in PR-WH, FISE consumes 1Mb TCAM storage, while the ACL-like structure consumes more than 72Mb. In the LB scenarios, the gain from FISE 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 these rules. Because LB requires fine-grained control on the flows by using such rules. LB could be achieved only based on the source address. However, it cannot fully leverage the advantages of multiple paths in Internet service provider networks [40]. Therefore, LB upon source and destination prefixes provides a better way to achieve fine-grained control on traffic.

In Fig. 14(e), we can see that, in the PR scenario, the SRAM space in FISE can be 1/40 as compared to that of the 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 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 VII-D, we show that the SRAM storage of FISE can be greatly reduced, due to the flexibility of SRAM.

Compression: Fig. 14(b) shows the TCAM space after compression. We also compress the 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 the ACL-like structure. In Fig. 14(f) depicts the SRAM space after compression. We can find that FISE can further compress SRAM after TCAM compression. For example, the SRAM storage of PR-WH is compressed to be less than 90K bits. This is because 1) the flexible mapping structure of FISE; and 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. 14(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 the ACL-like structure. However, to support non-homogeneous structure, the ACL-like structure must be physically divided, because most TCAM chips only support uniform entry width. In contrast, with FISE, we can flexibly and logically divide the table into two parts. Fig. 14(g) shows the SRAM space with non-homogeneous structure. We can see that, with non-homogeneous structure, FISE consumes much less SRAM storage than the ACL-like structure in all forwarding tables.

Compression with Non-Homogenous Structure: In Fig. 14(d) and 14(h), we apply non-homogeneous structure and compression techniques to FISE and the ACL-like structure. The resulting tables are the smallest among all tables. We can find that TCAM and SRAM spaces in FISE are much smaller than the ACL-like structure. The improvement in SRAM is very 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. 14(d), we see that in the PR scenario, although SPliT improves the 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 rules exist. In Fig. 14(h), we can observe 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 the ACL-like structure, 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 storage can be converted to monetary cost and power consumption, thus ISPs can save money/power [28].

2) Lookup and Update Operations: We show the lookup and update performance of FISE, and compare it with the ACL-like structure[4].

Lookup Speed: Fig. 13 presents the lookup speed without updates. We can observe 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 no packet loss. Note that the speed reaches the upper limit of the linecard we use, and it could 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. Fig. 15(a) shows the number of TCAM accesses per 100 updates in PR and LB scenarios. We can see that the number of TCAM accesses that FISE causes are three orders less than that of the ACL-like structure. For example, in the PR scenario, FISE causes 2-3 TCAM accesses per 100 updates while the ACL-like structure causes several thousands. This is because FISE stores much less information in TCAM.

Fig. 15(b) and 15(c) present the lookup speed, i.e., receiving rate on the traffic generator, of FISE with different update frequencies during 5 minutes. In Fig. 15(b), we can see that in the PR scenario, FISE has no influence on lookup while the ACL-like structure degrades the lookup speeds by 7% in the worst case. This is because FISE causes much fewer accesses to TCAM. In Fig. 15(c), we can find that in the LB scenario, FISE does influence the lookup speeds when there are 50,000 updates per second; however, the influence is still much smaller than the ACL-like structure.

We conclude that the 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, which can be finished within 1 second without influencing lookup. Note that, 1,301 is the number of times required to update rather than the update frequency. In our experiments, updates are asynchronous. We extract all the update sequences in the dataset and replay them with different frequencies, and then evaluate the effect of the update on the lookup. The maximum number of updates 1,301 is actually the worst case.

Fig. 16 shows cumulative probability of slowing down during lookup with different update frequency. We found that, as the update frequency increases, the lookup speed is more likely to slow down. When the lookup speed is less than 3 Gbps, the update speed will not be affected. For example, as shown in Figure 16(c), the update speed can reach 50,000 times per second.

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

Influence of SRAM Accesses During Update: FISE 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. 17, we show the number of accesses to SRAM with the incremental update and TD-Saturation(). We can find that for both PR and LB scenario, incremental updates cause much less accesses to SRAM. This is because during each update, TD-Saturation() has to reset all conflicted cells while incremental updates only have to reset a small part of them. For example, in the LB scenario, the incremental update causes 600 accesses in SRAM, while TD-Saturation() causes 10,814. In the PR scenario, the incremental update causes only 100 accesses; this is because the source table is composed of prefixes from U1 and U2, whose prefixes are disjoint except for two prefixes (240c::/28 and 240c:3::/32). Thus, updating a cell in TD-table brings almost none conflicted cells.

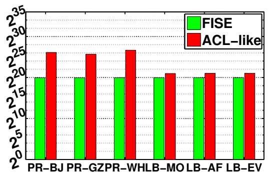
In Fig. 17b, we also show the computation time per 100 updates for both incremental update and TD-Saturation(). The result is similar with Fig. 17, because more accesses to SRAM indicate more cells that have to be computed.

In Fig. 18, 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. 18, 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.

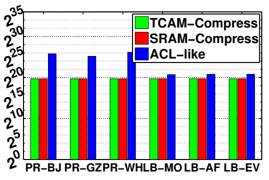
3) Evaluation Summary: Our evaluation shows that FISE can well accommodate 365,674 rules (this data is obtained from the PR-WH in the CERNET2 dataset.). For all scenarios, FISE consumes at most 1Mb TCAM and 7Mb SRAM space; this is because we fully eliminate the multiplicative effect in TCAM, and we can 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 FISE. Our experience shows that FISE can accommodate more rules. FISE achieves constant lookup time and works well with 50,000 updates per second, and we believe these do not hit the limit of FISE potential as well.

D. Simulation

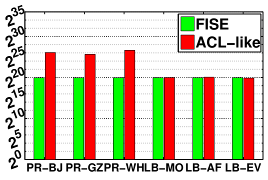
We set up simulations, to make a thorough evaluation on the storage size of FISE and the performance of fixed block deduplication. The rules are generated with the similar method mentioned in [41], which randomly selects the prefix pairs from the public routing tables to construct rules. To represent policies between organizations, e.g., enterprises [42], we randomly select K ASes from the



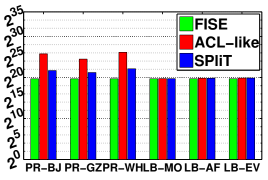
(a) TCAM with basic structure



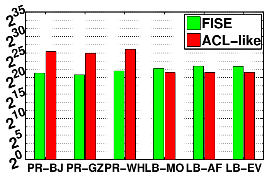
(b) TCAM after compression



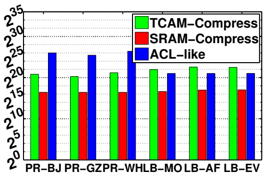
(c) TCAM with non-homogeneous



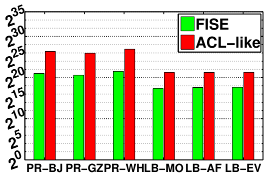
(d) TCAM after compression, with non-homo



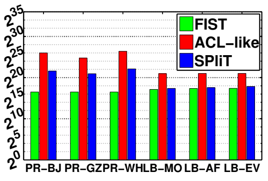
(e) SRAM with basic structure



(f) SRAM after compression

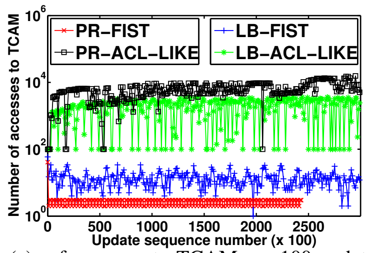


(g) SRAM with non-homogeneous

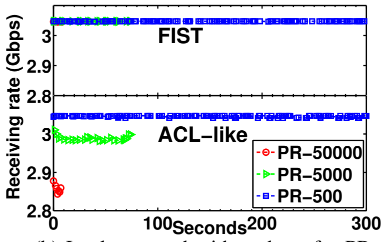


(h) SRAM after compression, with non-homo

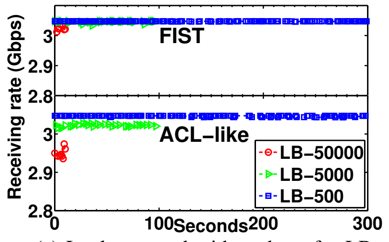
Fig. 14: Size of each forwarding table



(a) of accesses to TCAM per 100 updates

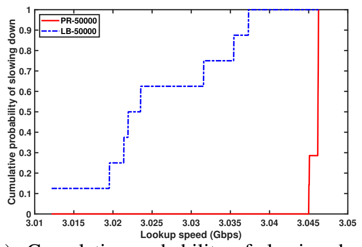


(b) Lookup speed with updates for PR

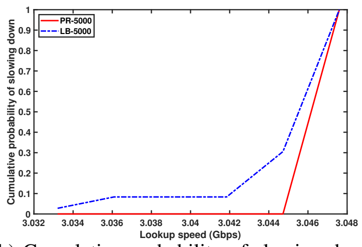


(c) Lookup speed with updates for LB

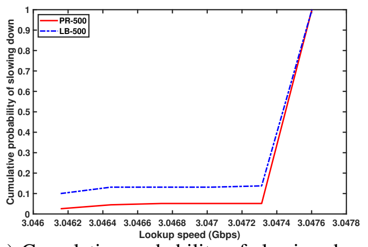
Fig. 15: Lookup speed with updates



(a) Cumulative probability of slowing down during lookup with 500 updates pre second

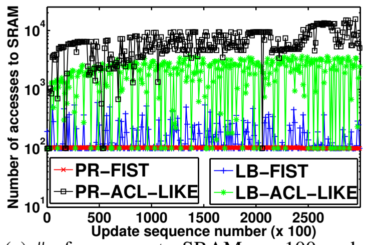


(b) Cumulative probability of slowing down during lookup with 5000 updates pre second

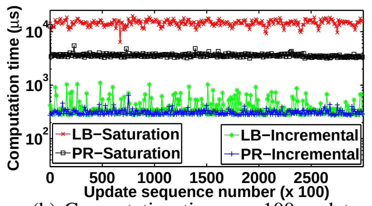


(c) Cumulative probability of slowing down during lookup with 50000 updates pre second

Fig. 16: Cumulative probability of slowing down during lookup with different update times pre second



(a) # of accesses to SRAM per 100 updates



(b) Computation time per 100 updates

Fig. 17: Comparison between incremental updates and TD-Saturation()

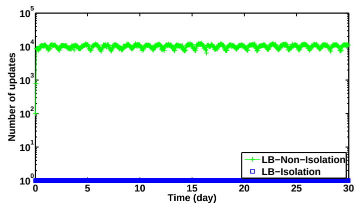
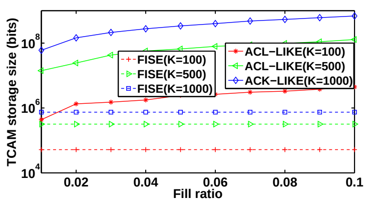


Fig. 18: Isolation VS. non-isolation

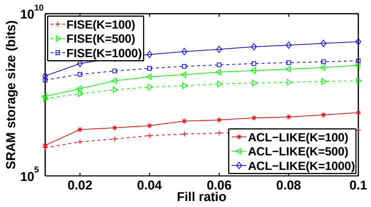
global ASes as the destination ASes, and another K ASes as the source ASes. Then, we choose χ destination and source AS pairs. For each pair, we randomly select a next hop between 1 and 255 (0 is reserved), such that all source prefixes in the source AS towards all destination prefixes will go through this next hop. We use the routviews prefix to AS mapping dataset for the simulation [43]. We define *fill ratio* as $\frac{\chi}{K^2}$, to evaluate the performance of FISE within different densities.

We use TCAM/SRAM size as the metric to compare FISE with fully compressed ACL-like structure, and use *deduplication ratio*, that is the SRAM size before deduplication divided by the SRAM size after deduplication, to evaluate the performance of deduplication.

Fig. 19(a) compares the TCAM storage size of FISE and the ACL-like structure at different fill ratio. We can see that no matter how large the fill ratio is, FISE consumes less TCAM. This is because FISE fully eliminates the “multiplication” factor. When the fill ratio increases, the TCAM storage of FISE remains roughly the same, while the TCAM storage of ACL-like structure increases rapidly. For example, when the fill ratio is 0.10 and K is 500, the ACL-like structure consumes 129Mb TCAM, while FISE only consumes 319Kb TCAM. This is because the ACL-



(a) TCAM storage size as a function of fill ratio



(b) SRAM storage size as a function of fill ratio

Fig. 19: Storage size as a function of fill ratio

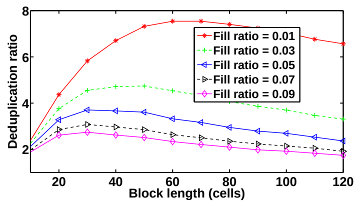


Fig. 20: Deduplication ratio as a function of block length with $K = 500$

like structure contains much redundancy.

Fig. 19(b) compares the SRAM size of FISE and the ACL-like structure. We can see when the fill ratio is small, FISE and ACL-like structure consume roughly the same SRAM, because FISE deduplicates most empty storage. When the fill ratio increases, both FISE and ACL-like structure consume more SRAM. However, the SRAM storage of FISE increases much slower, due to the flexible indexing structure.

In Fig. 20, we study the impact of block length. We set K to be 500 and evaluate the deduplication ratio with different block lengths. When the block length increases, the deduplication ratio increases first, due to the deflation of the catalog table; then it decreases, due to the inflation of the dictionary table. When the fill ratio is low, the block length that maximizes the deduplication ratio is large. The result agrees with the observation in Section V-A2. But more detailed work is out of the scope of this work.

VIII. Related Work

With increasing demands from users and ISPs for better and more flexible services, more routing policies are added into routers[13]. Many research works focus on new routing solutions, e.g., PBR [7], MTR [8], TCR [9] and recent software-defined networking (SDN). In Layer-3, more routing schemes make the routing decisions based on both source and destination addresses, such as policy routing [44], NIRA [45], customer-specific routing [15]. Recently, in IETF, many drafts have been proposed for Destination/Source routing [17]. Most of these solutions focus on re-designing the routing control plane. Our work re-designs the forwarding plane, which is orthogonal with them. The Lulea algorithm [46] also performed well in FIB lookup. However similar to other algorithm-based or caching-based methods, they do not have deterministic lookup speed [47, 48], which limit their practical deployment.

To support rich policies, the solutions can be divided into two broad categories: CAM-based and algorithmic solutions [49]. In this paper, we focus on CAM-based solutions. CAM-based, especially TCAM-based solutions are the de facto standard in industry. Most enterprise networks uses the ACL-like structure, which is ‘fat’ in TCAM and ‘thin’ in SRAM [14]. However, TCAM-based solutions are limited by its capacity [14]. In addition, TCAM is highly customized, so there are limited techniques we can use to compress it. The most popular technique is aggregation [12, 20, 50]. In [51, 52], the optimal two dimensional forwarding table compression is studied. However, in the extreme case, the compression ratio is only 20.0% on average [12]. In [12], a non-prefix approach that re-orders the ternary strings in prefixes to compress TCAM is investigated. Compared with them, we move the storage from TCAM to SRAM, thus 1) TCAM storage is reduced; and 2) More compression techniques can be used in SRAM.

HERMES [53] aims to minimize TCAM insertion times, and propose a novel idea that divides the TCAM table into shadow table and main table. However, lookups happen in both tables sequentially, thus HERMES belongs to ”fat” TCAM structure, and still contains the ”multiplication” factor in TCAM. Compared with HERMES, FISE wants to support the increasing number of policies and solve the limited capacity in TCAM, so divides the TCAM table into source table and destination table, and migrates storage to flexible SRAM. FISE changes the ”fat” TCAM structure to ”thin” TCAM structure, eliminating the ”multiplication” factor in TCAM.

There are studies proposing new structures to reduce the “multiplicative” effect in TCAM [14]. In [14], 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 [49]. A decision tree is a flowchart-like structure in which each internal node represents a “test” on an attribute, e.g., whether a coin flip comes up heads or tails, each branch represents the outcome of the test, and each leaf node represents a class label, i.e., decision taken after computing all attributes. The paths from root to leaf represent classification rules. Each time a packet arrives, the decision tree is traversed to find a leaf node, which stores a small number of rules. However, they suffer from non-deterministic performance and do not scale well [13]. Although we focus on CAM-based, we borrow the ideas from other non-CAM solutions. Bit-vector linear search [54] performs individual lookups in each dimension, and each dimension outputs a $O(n)$ length vector representing matched rules. By interseting bit-vectors, the algorithm computes the final result. Cross-producting [55] 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 can provide deterministic lookup speed. The Lulea algorithm also performs well in this situation, which can be applied in software-based routers. However, our FISE aims to provide hardware-based (i.e., TCAM-based) solution.

IX. Conclusion

In this paper, we have proposed a new forwarding table structure called FISE. Our focus is to accommodate the increasing number of policies in enterprise networks. Through the separation between TCAM and SRAM, FISE greatly reduces the TCAM storage and keeps fast lookup speed. By proposing an improved prefix tree called colored tree, we have designed the incremental updating algorithm, which can minimize the computation complexity and the number of accesses to memory. We have implemented the FISE-based forwarding table on a commercial router. Our design does not need any new devices and can be implemented on the existing hardware routers. We have also made comprehensive evaluations with the real implementation on a commercial router and the datasets from CERNET2. The results have shown that the performance of FISE is promising. This paper has focused on a 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. Besides, we can further reduce the size of the TD-table by leveraging the existing deduplication algorithms.

References

[1] Shu Yang, Mingwei Xu, Dan Wang, Gautier Bayzelon, and Jianping Wu. Scalable forwarding tables for supporting flexible policies in enterprise networks. In *Infocom, IEEE*, 2014.

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

[3] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. In *Proc. ACM SIGCOMM'12*, Helsinki, Finland, Aug 2012.

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

[5] Yinghui Zhang, Robert H. Deng, Gang Han, and Dong Zheng. Secure smart health with privacy-aware aggregate authentication and access control in internet of things. *Journal of Network and Computer Applications*, 123:89 – 100, 2018.

[6] Keke Gai, Meikang Qiu, and Xiaotong Sun. A survey on fintech. *Journal of Network and Computer Applications*, 103:262 – 273, 2018.

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

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

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

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

[11] Theophilus Benson, Aditya Akella, and David A. Maltz. Mining policies from enterprise network configuration. In *Proc. ACM IMC'09*, Chicago, IL, Nov 2009.

[12] C.R. Meiners, A.X. 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.

[13] Yadi Ma and Suman 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] Chad R. Meiners, Alex X. Liu, Eric Torng, and Jignesh Patel. Split: Optimizing space, power, and throughput for tcam-based classification. In *Proc. ACM/IEEE ANCS'11*, Brooklyn, NY, Oct 2011.

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

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

[17] D. Lamparter. Destination/source routing. Internet Draft, Oct 2014. draft-lamparter-rtgwg-dst-src-routing-00.txt.

[18] Anindya Tahsin Prodhan, Rajkumar Das, Humayun Kabir, and Gholamali C. Shoja. Ttl based routing in opportunistic networks. *Journal of Network and Computer Applications*, 34(5):1660 – 1670, 2011.

[19] Shu Yang, Laizhong Cui, Xinhao Deng, Qi Li, Yulei Wu, Mingwei Xu, Dan Wang, and Jianping Wu. Two dimensional router: Design and implementation. Technical report, Tsinghua University, Aug 2019. https://arxiv.org/pdf/1908.04374.pdf.

[20] A.X. Liu, C.R. 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.

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

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

[23] Junghwan Kim, Myeong-Cheol Ko, Hyun-Kyu Kang, and Jinsoo Kim. A hybrid ip forwarding engine with high performance and low power. In *Proc. ICCSA'09*, Seoul, Korea, Jun 2009.

[24] Richard P. Draves, Christopher King, Srinivasan Venkatachary, and Brian N. Zill. Constructing optimal ip routing tables. In *Proc. IEEE INFOCOM'99*, New York, NY, March 1999.

[25] Nagapramod Mandagere, Pin Zhou, Mark A Smith, and Sandeep Uttamchandani. Demystifying data deduplication. In *Proc. ACM/IFIP/USENIX Companion'08*, Leuven, Belgium, Dec 2008.

[26] Tania Mishra and Sartaj Sahni. Duos - simple dual tcam architecture for routing tables with incremental update. In *Proc. IEEE ISCC'10*, Riccione, Italy, Jun 2010.

[27] D. Geer. Reducing the storage burden via data deduplication. *Computer*, 41(12):15 –17, 2008.

[28] Shu Yang, Dan Wang, Mingwei Xu, and Jianping Wu. Two dimensional router: Design and implementation. Technical report, Tsinghua University, Aug 2012. http://www.wdklife.com/tech.pdf.

[29] Ju Hyoung Mun and Hyesook Lim. Cache sharing using bloom filters in named data networking. *Journal of Network and Computer Applications*, 90:74 – 82, 2017.

[30] João Luís Sobrinho, Laurent Vanbever, Franck Le, and Jennifer Rexford. Distributed route aggregation on the global network. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, New York, NY, USA, Dec 2014.

[31] S. Maity, P. Bera, and S. K. Ghosh. Policy based acl configuration synthesis in enterprise networks: A formal approach. In *2012 International Symposium on Electronic System Design (ISED)*, Kolkata, India, Dec 2012.

[32] Benjamin Zhu, Kai Li, and Hugo Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proc. USENIX FAST'08*, San Jose, California, Feb 2008.

[33] Biplob Debnath, Sudipta Sengupta, Jin Li, David J Lilja, and David HC Du. Bloomflash: Bloom filter on flash-based storage. In *2011 31st International Conference on Distributed Computing Systems*, pages 635–644. IEEE, 2011.

[34] Openssl. http://www.openssl.org.

[35] Mohamed Khalil-Hani, Vishnu 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.

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

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

[38] Jianping Wu, Jessie Hui Wang, and Jiahai Yang. Cngi-cernet2: an ipv6 deployment in china. *ACM SIGCOMM Computer Communication Review*, 41(2):48–52, 2011.

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

[40] Juan Antonio Cordero. Multi-path tcp performance evaluation in dual-homed (wired/wireless) devices. *Journal of Network and Computer Applications*, 70:131 – 139, 2016.

[41] Florin Baboescu, Priyank Warkhede, Subhash Suri, and George Varghese. Fast packet classification for two-dimensional conflict-free filters. *Comput. Netw.*, 50(11):1831–1842, 2006.

[42] Patrick Agyapong and Marvin Sirbu. The economic implications of edge-directed routing: a network operator's perspective. In *Proc. INTERNET 2012*, Venice, Italy, Jun 2012.

[43] CAIDA. Routevies prefix to as mapping dataset. http://www.caida.org/data/routing/routeviews-prefix2as.xml.

[44] Arun Seehra, Jad Naous, Michael Walfish, David Maziššres, Antonio Nicolosi, and Scott Shenker. A policy framework for the future internet. In *Proc. ACM HotNets'09*, New York, NY, Oct 2009.

[45] Xiaowei Yang, David Clark, and Arthur W. Berger. Nira: A new inter-domain routing architecture. *IEEE/ACM TRANSACTIONS ON NETWORKING*, 15, 2007.

[46] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. Small forwarding tables for fast routing lookups. In *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, New York, NY, USA, Oct 1997.

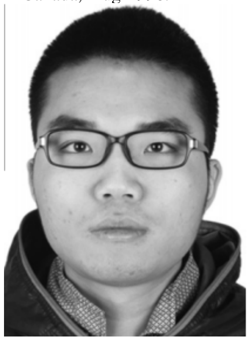
[47] Dr. Zahid Ullah, Manish Jaiswal, and Ray C.C. Cheung. Z-tcam: An sram-based architecture for tcam. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(2):402–406, 2015.

[48] Andrey Belenkiy and Necdet Uzun. Deterministic ip table lookup at wire speed. In *Proc. INET99*, 1999.

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

[50] Shui Yu, Meng Liu, Wanchun Dou, Xiting Liu, and Sanming Zhou. Networking for big data: A survey. *IEEE Communications Surveys Tutorials*, 19(1):531–549, 2017.

[51] Subhash Suri, Tuomas Sandholm, and Priyank Warkhede. Compressing two-dimensional routing tables. *Algorithmica*, 35:287–300, 2003.
[52] Bohao Feng, Hongke Zhang, Huachun Zhou, and Shui Yu. Locator/identifier split networking: A promising future internet architecture. *IEEE Communications Surveys Tutorials*, 19(4):2927–2948, 2017.
[53] Huan Chen and Theophilus Benson. Hermes: Providing tight control over high-performance sdn switches. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 283–295. ACM, 2017.
[54] 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.
[55] 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.

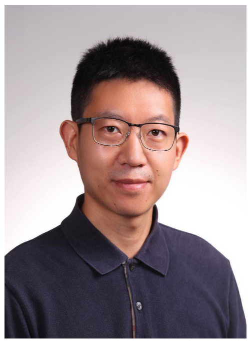


Shu Yang received his B.Sc. degree from Beijing University of Posts and Telecommunications and Ph.D. degree from Tsinghua University. His research interest includes network architecture and high performance router.

Laizhong Cui received the B.S. degree from Jilin University, Changchun, China, in 2007 and Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2012. He is currently an associate professor in the College of Computer Science and Software Engineering at Shenzhen University, China. He led the projects of the National Key Research and Development Program of China and the National Natural Science Foundation, and several projects of Guangdong Province and Shenzhen City. His research interests include future Internet architecture, edge computing, big data, IoT, computational intelligence, software-defined network and machine learning.

Xinhao Deng received his B.Sc. degree from Hangzhou Dianzi University. He is a master candidate in Institute for Network Sciences and Cyberspace, Tsinghua University. His research interest includes network functions virtualization and high performance router.

Qi Li received the PhD degree from Tsinghua University. Now he is an associate professor of Institute for Network Sciences and Cyberspace, Tsinghua University. He has ever worked in ETH Zurich, the University of Texas at San Antonio, The Chinese University of Hong Kong, and Chinese Academy of Sciences. His research interests include network and system security, particularly in Internet and cloud security, mobile security, and big data security. He is currently an editorial board member of IEEE TDSC and ACM DTRAP.



Yulei Wu received the Ph.D. degree in Computing and Mathematics and the B.Sc. degree (1st Class Hons.) in Computer Science from the University of Bradford, United Kingdom, in 2010 and 2006, respectively. He is a Senior Lecturer in the Department of Computer Science with the University of Exeter, United Kingdom. His expertise is on networking and his main research interests include intelligent networking technologies, network slicing and softwarization, etc. He is an Editor of IEEE Transactions on Network and Service Management, Elsevier Computer Networks and IEEE Access. He contributes to major conferences on networking as various roles including a Steering Committee Chair, a General Chair, a Program Chair, and a Technical Program Committee Member. He is a Senior Member of the IEEE, and a Fellow of the HEA (Higher Education Academy).

Mingwei Xu received his B.Sc. degree and Ph.D. degree from Tsinghua University. He is a full professor with the Department of Computer Science. His research interest includes computer network architecture, high-speed router architecture. He is a member of the IEEE.

Dan Wang received his B. Sc from Peking University, M. Sc from Case Western Reserve University and Ph. D. from Simon Fraser University. He is an Associate Professor of Department of Computing. His research interest includes Sensor Networks, Internet Routing. He is a member of the IEEE.

Jianping Wu received his B.S., M.S., and Ph.D. from Tsinghua University. He is a Full professor and director of Network Research Center, Ph.D. Supervisor of Department of Computer Science, Tsinghua University. His research interests include next generation Internet, IPv6 deployment and technologies, Internet protocol design and engineering. He is an IEEE fellow.