H-SOFT: A Heuristic Storage Space Optimisation Algorithm for Flow Table of OpenFlow

Jingguo Ge[1], Zhi Chen[1,2], Yulei Wu[1], Yuepeng E[1]

[1]Computer Network Information Center, Chinese Academy of Sciences, Beijing, 100190, China

[2]University of Chinese Academy of Sciences, Beijing, 100049, China

Abstract

OpenFlow has become the key standard and technology for software defined networking which has been widely adopted in various environments. However, the global deployment of OpenFlow encountered several issues, such as the increasing number of fields and complex structure of flow entries, making the size of flow table in OpenFlow switches explosively grows which results in hardware implementation difficulty. To this end, this paper presents the modelling on the minimisation for storage space of flow table, and proposes a Heuristic Storage space Optimisation algorithm for Flow Table (H-SOFT) to solve this optimisation problem. The H-SOFT algorithm degrades the complex and high-dimensional fields of a flow table into multiple flow tables with simple and low-dimensional fields based on the co-existence and conflict relationships among fields to release the un-used storage space due to blank fields. Extensive simulation experiments demonstrate that the H-SOFT algorithm can effectively reduce the storage space of flow table. In particular, with frequent updates on flow entries, the storage space compression rate of flow table is stable and can achieve at ~70%. Moreover, in comparison with the optimal solution, the H-SOFT algorithm can achieve the similar compression rate with much lower execution time.

Keywords: OpenFlow, Flow table, Storage space optimisation, SDN

1. Introduction

With rapid expansion of network scales and constant enrichment of applications, the structure of current Internet is more complex and rigid and the control capability of the Internet is becoming increasingly weak [1]. Due to the lack of interoperation between Internet applications and underlying communication networks, applications cannot obtain the status of networks, and the network cannot be aware of the requirements of applications based on which for its configuration. The Software Defined Network (SDN) [2] is an innovative network architecture which separates the control plane and forwarding plane with the characteristics of standardised interface, global view, flexible control and open programmability. SDN can provide an open platform for the innovation of networks and applications, which has a revolutionary impact on future networks.

OpenFlow [1, 3] has become the standard and key technology for the implementation of SDN and has been widely adopted in various environments, e.g., data center networks, to facilitate the traffic engineering and improve network performance. OpenFlow technology is now demanding the deployment in large-scale, even global-scale, networks to provide the opportunities for researchers to test and evaluate their innovative designs. However, due to the explosive growth of network bandwidth, the diversification of network applications, the development of new network architectures, as well as flexible and efficient network management and control requirements, the size and structure of flow tables in an OpenFlow switch are facing new challenges in global-scale deployment.

- *The size of flow table in OpenFlow switch will grow explosively*. Current IPv4 BGP routing table has more than 440,000 entries which are only based on one field, i.e., IP destination. With a fine-grained control granularity, one data flow may correspond to more than one flow entries in the flow table, which could result in explosive expansion on the size of flow table in the OpenFlow switch.
- *The number of fields will constantly increase and the structure of flow table will become more complex*. OpenFlow switch specification version 1.3.0 (OpenFlow1.3) defines 40 fields in the match field [4], while in the future, the specification may extend to include

new fields, e.g., content ID and service ID, due to the deployment of new Internet architectures and applications such as content-oriented architecture and service-oriented architecture [5]. In addition, the network security, Quality-of-Service, load balancing and scalability are required to achieve flexible and efficient network control and management. Therefore, with increasing extension on the functionality of OpenFlow switches, the structure of flow table is becoming more complex.

Traditional routing table only matches the field of IPv4 destination or IPv6 destination. The field length of routing entries mainly occupies 32 bits. The entire length of 40 fields of match field defined in OpenFlow1.3 is 1227 bits, because the match field contains many protocol fields including the VLAN, IPv4, IPv6, MPLS, TCP, UDP, SCTP, ICMPv4, ICMPv6, and ARP. However, some protocol fields cannot exist simultaneously, for example, UDP field appears when the TCP and SCTP fields are invalid. In other words, the number of valid fields in the match filed of entries in flow table must be less than 40. When a flow table stores all fields, each flow entry occupies 1227 bits, resulting in a high-dimensional sparse storage structure and storage space wastage. Moreover, the explosive growth of flow table size, the increasing number of fields, and the complex structure of flow entries make the storage space wastage more serious.

Target at the challenging problem of reducing the storage space for flow table, this paper makes the following contributions:

- The modelling on the minimisation of storage space for flow table in an OpenFlow switch is presented. To solve this optimisation problem, a Heuristic Storage space Optimization algorithm for Flow Table (H-SOFT) is then proposed. The H-SOFT algorithm degrades the complex fields in match field of a flow table into multiple flow tables with simple fields, called sub-flow tables. The storage compression is achieved by assigning each flow entry into those sub-flow tables.
- The H-SOFT algorithm supports frequently adding new flow entries into flow tables by adjusting the valid fields in each sub-flow table if the number of new flow entries to be added exceeds a predefined threshold, to optimise the storage space while still maintain

the efficiency of the algorithm.

- Extensive simulation experiments are conducted to verify the effectiveness of the proposed H-SOFT algorithm. By virtue of the proposed algorithm, the storage space compression rate of flow table is stable and can achieve at ~70%. Moreover, in comparison with optimal solution, the H-SOFT algorithm can achieve the similar compression rate with much lower execution time.

The rest of this paper is organised as follows. Section 2 shows the related work. Section 3 analyses the structure of flow table and presents the modelling of storage space minimisation. The H-SOFT algorithm is presented in Section 4. Section 5 validates the effectiveness of the proposed algorithm through extensive simulation experiments. Finally, Section 6 concludes this study.

2. Related Work

OpenFlow is a new network exchange model proposed by Stanford University in 2007 to support innovative network research. It separates the control plane and forwarding plane in the network. The network programmability is achieved through open and dynamical updating on the flow entries which are stored in the flow table of an OpenFlow switch. The data packets can be forwarded according to the flow entries, while the controller is responsible for their generation, maintenance, and configuration.

In IP networks, packet classification is based on the key fields of packet header, which makes the network equipments take differentiated actions for various network services. In OpenFlow networks, the equipments manage and control the network based on flow tables. Thus, the existing studies, such as RFC [6, 7], Grid of Trie [8], HiCuts [9] and HyperCuts [10], on packet classification could facilitate the research on efficient lookup of flow tables in OpenFlow switches. RFC algorithm could achieve high network throughput, but it demanded large storage space, which cannot be applicable for classification of large rule base; and thus RFC is not suitable for the lookup in large and complex flow tables of OpenFlow switches. Grid of Trie algorithm was based on the hierarchical binary search tree, which requires low storage space to improve the search performance but with high complexity on tree update; and thus Grid of Trie cannot be adopted for flow table lookup due to the high frequency of update operation on the entries of flow table. HiCuts and HyperCuts algorithms were used to solve the problem of low-dimensional packet classification. However, the entries in flow table contains multi-dimensional fields, so that the use of HiCuts and HyperCuts for flow table lookup could cause the explosion of storage space and low efficiency of lookup operation.

The flow tables in OpenFlow switches have similar characteristics of traditional routing table and Access Control List (ACL). The routing table uses the destination IP addresses as the matching field where multiple entries can be aggregated into a single entry [11]. The authors in [12] proposed two sub-optimal Forwarding Information Base (FIB) compression algorithms based on the proposed Election and Representative (EAR) algorithm. The two suboptimal algorithms preserved the structure information, and supported fast incremental updates while reducing computational complexity. The performance merits come at the cost of only 1.5% loss in compression ratio compared with that in theoretically optimised ratio. Karpilovsky et al. [13] presented an incrementally deployable memory management system that can reduce the associated router state by up to 70%. The system coalesced prefixes to reduce storage consumption and can be deployed locally on each router or centrally on a server for routing. However, the match field in OpenFlow switch contains the fields spanning from Layer 1 to Layer 4, in which many fields cannot be aggregated. Zeng and Yang [14] proposed to adopt the cross-coverage and inclusion relationships between one statement and multiple statements and between multiple statements and multiple statements to reduce the number of ACL. Daly, Liu and Torng [15] proposed Diplomat algorithm to compress the ACL by transforming high-dimensional matching targets into low-dimensional ones through dividing the original matching targets into a series of hyper-planes. Diplomat can achieve an average improvement ratio of 30.6% over firewall compressor. The authors in [16] proposed ACL compressor to handle one-dimensional and multi-dimensional ACL, which can significantly reduce the number of rules by 50.22% in an ACL while maintaining the same semantics. Curtis et al. [17] pointed out that the OpenFlow technology is based on data flow instead of the destination address, resulting in the more entries included in flow table of OpenFlow switches than those in traditional routing table under the same network traffic conditions. In other words, one data flow may correspond to multiple flow entries to achieve the control over different granularity. The control granularity could be broken if the storage space optimisation is achieved based on the relationship among flow entries. Therefore, the ACL storage optimisation techniques are not suitable to achieve the same purpose for flow table of OpenFlow switches.

According to the characteristics of flow table in OpenFlow, Curtis et al. [17] proposed the algorithm to reduce the number of flow entries by revising the DevoFlow model of OpenFlow, but this algorithm has low extensibility. The authors in [18] presented the DIFANE model which combines the active and passive installation scheme of flow tables to keep traffic in the forwarding plane and reduce the message exchange between the controller and switches, and thus lower the traffic load of the controller. The reduction of the number of flow entries can be achieved by applying the wildcard in the fields, while the existence of the wildcard entry affects the use of hash-based lookup on a flow table [19]. In addition, OpenFlow1.3 proposed a multi-level flow tables with pipeline processing to compress the storage space. However, the overhead of actual flow table storage is related to the design of multi-level flow table structure which has not been given in the specification.

Traditional packet classification algorithms are mainly used for low-dimensional and small-scale rule base and it is not suitable for the rule base with the characteristic of dynamic update. The storage space optimisation algorithms developed for routing table and ACL are based on their unique characteristics and are thus not extensible to be used for flow table of OpenFlow. Moreover, the modification on OpenFlow models can not solve the issue of storage space wastage of flow tables. In contrast, the H-SOFT algorithm proposed in this paper analyses the structure of flow table defined in OpenFlow1.3 and provides the modelling on the storage space optimisation particularly for flow table of OpenFlow.

3. The mathematical model

This paper focuses on the OpenFlow1.3 [4] and targets on the optimisation of storage space for flow table in an OpenFlow switch. Table 1 shows the 40 fields including 13 required fields and 27 optional fields in the match field of flow table defined in OpenFlow1.3.

Let $T = \{R_i \mid 1 \le i \le n\}$ denote the n flow entries in the flow table of an OpenFlow switch, and $D = \{F_i \mid 1 \le i \le d\}$ represent the d fields in the match field of R_i. Thus, the Sets T and D can be adopted to depict a flow table in an OpenFlow switch.

Definition 1. The co-existence relationship, Coe, of any two fields, F_i and F_j, in D: $Coe = \{(F_i, F_j) \mid F_i \in D, F_j \in D, F_j$ must exist if F_i exists and F_j must not exist if F_i does not exist$\}$[1]. Coe possesses the characteristics of reflexiveness, symmetry, and transitive relation. For example, $(IPv4\_SRC, IPv4\_SRC) \in Coe$, $(IPv6\_SRC, IPv6\_DST) \in Coe$, and $(IPv6\_DST, IPv6\_SRC) \in Coe$.

Definition 2. The conflict relationship, Col, of any two fields, F_i and F_j, in D: $Col = \{(F_i, F_j) \mid F_i \in D, F_j \in D, F_i$ and F_j cannot coexist$\}$. Col has the characteristics of anti-reflexiveness and symmetry. For example, $(TCP\_SRC, TCP\_SRC) \notin Col$, $(TCP\_SRC, UDP\_DST) \in Col$, and $(UDP\_DST, TCP\_SRC) \in Col$.

Let us use the 13 required fields in the match field of flow table in OpenFlow1.3 as an example to illustrate the Definition 1 and Definition 2, in which $D = D'$ where $D' = \{IN\_PORT, ETH\_DST, ETH\_SRC, ETH\_TYPE, IP\_PROTO, IPV4\_SRC, IPV4\_DST, IPV6\_SRC, IPV6\_DST, TCP\_SRC, TCP\_DST, UDP\_SRC, UDP\_DST\}$ [2]. According to Definition 1, the Coe of each field in D' can be depicted as a two-dimensional matrix, M_{Coe}, shown in Eq. (1), and the Col of each field in D' represented by M_{Col} can be obtained based on Definition 2 and can be given by Eq. (2). In Eqs. (1) and (2), $M_{Coe}[i][j] = 1$ denotes the Coe relationship between the fields F_i and F_j in D', and $M_{Coe}[i][j] = 0$ represents these two fields do not have such relationship. Similarly, $M_{Col}[i][j] = 1$ and $M_{Col}[i][j] = 0$ can reflect that the fields F_i and F_j in D' possess the Col and non-Col relationships, respectively.

[1] The fields F_i and F_j represent the fields appearing in one transmission unit.
[2] The order of items in the Set D' cannot be changed.

$$M_{coe} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \qquad (1)$$

$$M_{col} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \qquad (2)$$

Recall that the flow table can be represented by the two Sets $T = \{R_i \mid 1 \le i \le n\}$ and $D = \{F_i \mid 1 \le i \le d\}$. To decrease the storage space, each flow table can be partitioned into k sub-flow tables according to the Coe and Col relationships, given by Definitions 1 and 2, between any two fields, where the i-th sub-flow table, $1 \le i \le k$, including u_i flow entries and v_i fields in match field can be denoted by $T_i = \{R_i \mid 1 \le i \le u_i\}$ and $D_i = \{F_i \mid 1 \le i \le v_i\}$ where $T_i \subseteq T$ and $D_i \subseteq D$; moreover, $T_i \cap T_j = \varnothing$ for $\forall i, j$ and $i \ne j$, $i \in [1, u_i]$, and $j \in [1, u_j]$. Such a partition can decrease the number of fields in each sub-flow table in comparison with that all fields are stored in a single flow table. It is worth noting that the number of sub-flow tables cannot exceed a threshold K, i.e., $k \le K$.

Let $Bv_i = [b_{i1}, b_{i2}, \ldots, b_{id}]$ represent a d-dimensional 0-1 vector to store the valid fields in the i-th sub-flow table, where b_{ij} is set to be 1 if the j-th bit has a valid field F_i, otherwise $b_{ij} = 0$. Let $W = \{w_i \mid 1 \le i \le d\}$ denote the width in bits of F_i in D, and $W_i = \sum_1^d w_j \times b_{ij}$ represent the storage space for a given flow entry in the i-th sub-flow table, and thus $u_i \times W_i = u_i \times \sum_i^d w_j \times b_{ij}$, $1 \le i \le k$, denotes the total storage space for all flow entries in the i-th sub-flow table. Let N denote the total number of flow entries in all sub-flow tables. N can be given by

$$N = \sum_{i=1}^{k} u_i \tag{3}$$

To minimise the storage space, we can formulate the optimisation problem as follows:

$$\min \ p = 1 - \frac{f(k)}{f(1)} = 1 - \frac{\sum_{i=1}^{k} u_i \times \sum_{j=1}^{d} w_j \times b_{ij}}{\sum_{i=1}^{k} u_i \times \sum_{j=1}^{d} w_j} \tag{4}$$

$$\text{s.t.} \quad 1 \le k \le K \tag{5}$$

$$0 < W_i \le \sum_{j=1}^{d} w_j \ \text{ and } \ 1 \le i \le k \tag{6}$$

where p denotes the compression rate of storage space for flow table, $f(k)$ represents the required storage space for k sub-flow tables formed by the partition of a single flow table in an OpenFlow switch, and $f(1)$ is the required storage space for one flow table containing all flow entries. With the frequent updating of flow entries in the flow table of an OpenFlow switch, $f(1)$ is increasingly changed, and thus the only use of $f(k)$ cannot validate the effectiveness of the proposed H-SOFT algorithm given in Section 4. Therefore, this paper introduces a key parameter, p, where the larger of p, the more effective of the proposed algorithm.

4. An heuristic storage space optimisation algorithm for flow table in an OpenFlow switch (H-SOFT)

At the initialisation stage, a part of key flow entries will be sent to the flow table in OpenFlow switch by the controller in advance, to lower the frequency of message exchange between the controller and switches. The Heuristic Storage space Optimisation algorithm for Flow Table (H-SOFT) leverages the M_{Coe} and M_{Col} matrices, representing the Coe and Col relationships between any two fields of flow table, to obtain k, where $1 \le k \le K$, initial sub-flow tables, each of which is assigned with the flow entries sent by the controller. With the changing status of network flows, the flow entries in flow table are increasingly changed as well, e.g., adding new flow entries, deleting and updating existing flow entries, etc. However, the fields in match filed will not get affected and changed with the updating action [4]. Therefore, this paper focuses on adding and deleting flow entries and presents the

H-SOFT in the Algorithm 1.

Algorithm 1: The efficient storage space optimisation algorithm for flow table in an OpenFlow switch: $H-SOFT(\{s\},\{e\},single\_ft,sub\_ft,M_{Coe},M_{Col},n,K)$

Input: Initial flow entries: $\{s\}$; Flow entries to be added: $\{e\}$; Single flow table: $single\_ft$; Sub-flow table: sub_ft; Relationship matrix: M_{Coe} and M_{Col}; Dynamic tuning threshold: n; Maximum number of sub-flow tables: K;
Output: Updated sub-flow table: sub_ft

```
1:  Begin
2:      Initial $sub\_ft$ can be created by $InitiationClusters(single\_ft,M_{Coe},M_{Col},K)$;
3:      For $s \in \{s\}$
4:          Flow entries can be added into the $sub\_ft$ using $AddEntry(s,sub\_ft)$;
5:      EndFor
6:      For $e \in \{e\}$
7:          Add flow entry $e$ into $sub\_ft$ using $AddEntry(e,sub\_ft)$;
8:          $num\_of\_updates++$;
9:          If $num\_of\_updates \geq n$
10:             $sub\_ft$ = $SplitCluster(sub\_ft)$;
11:             $num\_of\_sub\_flow\_tables++$;
12:             $num\_of\_updates = 0$;
13:         EndIf
14:         If $num\_of\_sub\_flow\_tables > K$
15:             $sub\_ft = MergeCluster(sub\_ft)$;
16:         EndIf
17:     EndFor
18:     Return $sub\_ft$;
19: End
```

In the Algorithm 1, the $InitiationClusters()$ function in Algorithm 2 is used to form k initial sub-flow tables, into which the $AddEntry()$ function in Algorithm 5 is then adopted to add the initial flow entries, $\{s\}$. The flow entry e can be added into one of k sub-flow tables using $AddEntry()$ function again. If the number of flow entries to be changed exceeds the dynamic tuning threshold, n, the $SplitCluster()$ function in Algorithm 3 can be used to split k sub-flow tables to $(k+1)$ sub-flow tables. If the number of sub-flow tables is greater than the pre-defined threshold, K, the $MergeCluster()$ function in Algorithm 4 is then used to merge all sub-flow tables into K sub-flow tables. The interoperate between the functions $SplitCluster()$ and $MergeCluster()$ results in the optimisation of storage space of flow table.

Algorithm 2: Obtain k initial sub-flow tables: $InitiationClusters(sub\_ft,M_{Coe},M_{Col},K)$

Input: Sub-flow tables: sub_ft; Relationship matrix: M_{Coe} and M_{Col}; Maximum number of sub-flow tables: K;
Output: The initial sub-flow tables: sub_ft

```
1:  Begin
2:      While $num\_of\_sub\_flow\_tables < K$
3:          Select a sub-flow table $c$ with the maximum bit size of its fields in Match Field;
4:          Split $c$ into two flow tables $c'$ and $c''$ based on Matrices $M_{Coe}$ and $M_{Col}$,
            to allow $c'$ and $c''$ to have minimum bit size of their fields;
5:          Remove the sub-flow table $c$ from $sub\_ft$;
6:          If $c' \notin sub\_ft$
7:              Add the flow table $c'$ into $sub\_ft$;
8:          EndIf
9:          If $c'' \notin sub\_ft$
10:             Add the flow table $c''$ into $sub\_ft$;
11:         EndIf
12:         If $num\_of\_sub\_flow\_tabls \geq K$
13:             return $sub\_ft$;
14:         EndIf
15:     EndWhile
16:     return $sub\_ft$;
17: End
```

The Algorithm 2 splits one flow table containing all flow entries into k sub-flow tables. Specifically, the algorithm first selects a sub-flow table, c, with the maximum storage space of the fields, i.e., $\max(W_i)$, $1 \leq i \leq k$, and then do the splitting on c to obtain two flow tables, c' and c'', with the maximum reduction on the storage space of the fields. The new sub-flow tables is updated by deleting c and adding c' and c''.

Algorithm 3: Split sub-flow tables: $SplitClusters(sub\_ft)$

Input: Sub-flow tables: sub_ft;
Output: The new sub-flow tables: sub_ft

```
1:  Begin
2:      Choose sub-flow table $c$ with the maximum storage space wastage
3:      For $i$ = 0 to $num\_of\_fields\_in\_sub\_flow\_table\_c - 1$
4:          For $j$ = 0 to $num\_of\_fields\_in\_sub\_flow\_table\_c - 1$
5:              $S'_j$ = Storage space wastage of the $j$-th field;
6:          EndFor
7:          Arrange $S'_j$ in the descending order and store them in $S_{i,j}$;
8:      EndFor
9:      For $i$ = 1 to $num\_of\_elements\_in\_S_{i,j}$
10:         $Wastage_i \leftarrow$ Sum of storage space wastage from the first to the $i$-th fields;
11:         If $Wastage_i < Wastage_{i-1}$
12:             Break;
13:         EndIf
14:     EndFor
15:     Split $c$ into $c'$ and $c''$ based on the value of $i$;
16:     $c'$ contains all fields in the sub-flow table $c$;
17:     $c''$ contains the fields in the $c$ except for the first $(i-1)$ number of fields in $S_{i,j}$;
18:     Remove $c$ from $sub\_ft$;
19:     Add $c'$ and $c''$ into $sub\_ft$;
20:     Return $sub\_ft$;
21: End
```

In Algorithm 3, the k sub-flow tables are split into $(k+1)$ sub-flow tables. The algorithm chooses the sub-flow table c with the maximum storage space wastage, and then calculates the storage space wastage of the j-th field in c and stores them in the descending order in $S_{i,j}$. Sub-flow table, c, is split into flow tables, c' and c'', based on the value of parameter i which is determined by $Wastage_i < Wastage_{i-1}$. In addition, the flow entries in flow table c are then added into the new flow table c' or c'' using $AddEntry()$ function. If the number of sub-flow tables exceeds the threshold K, the Algorithm 4 is adopted to merge $(k+1)$ flow tables into k flow tables.

Algorithm 4: Merge sub-flow tables: $MergeClusters(sub\_ft)$

Input: Sub-flow tables: sub_ft
Output: The new sub-flow tables: sub_ft

```
1:  Begin
2:      For $i = 0$ to $(num\_of\_sub\_flow\_tables - 1)$
3:          For $j = i$ to $num\_of\_sub\_flow\_tables$
4:              $c' \leftarrow$ the $i$-th sub-flow table;
5:              $c'' \leftarrow$ the $j$-th sub-flow table;
6:              $c \leftarrow$ contain all flow entries of flow tables $c'$ and $c''$;
7:              $\Delta\_storage\_space = N_c W_c - N_{c'} W_{c'} - N_{c''} W_{c''}$;
            EndFor
8:          $increased\_storage\_space = \min(\Delta\_storage\_space)$;
9:          Select $c'$ and $c''$ based on $increased\_storage\_space$ to get $c$;
10:         For each entry $e$ in $c'$ and $c''$:
11:             $AddEntry(e,c)$;
12:         EndFor
13:     EndFor
14:     Remove the flow tables $c'$ and $c''$ from $sub\_ft$;
15:     Add the flow table $c$ to $sub\_ft$;
16:     Return $sub\_ft$;
17: End
```

The Algorithm 5 focuses on the selection of an appropriate sub-flow table, into which the flow entry e is added. Specifically, for each of k sub-flow tables, do the subtraction between each bit of the vector Bv and the corresponding bit in flow entry e, and obtain the summation of k absolute values, abs_dist, and non-absolute value, $dist$. It is worth noting that the fields in flow entry e are converted to 0 and 1 based on their validity value. If abs_dist equals to $dist$, the flow entry, e, can be added into this sub-flow table. The flow entry, e, is finally added into a sub-flow table with the minimum storage space.

Algorithm 5: Add a flow entry into a sub-flow table: $AddEntry(e, sub\_ft)$

Input: Flow entry to be added: e; Sub-flow table: sub_ft;
Output: The sub-flow table c into which the flow entry is added

```
1:  Begin
2:      $chose[k] = \{0\}$;
3:      $num = 0$;
4:      For $c \in sub\_ft$
5:          $num++$;
6:          Obtain the $Bv$ of sub-flow table $c$;
7:          Convert the value of fields in flow entry $e$ to 0 or 1 and store the value in $Bv\_e$;
8:          For $i$ in $length\_of\_vector\_Bv$
9:              $abs\_dist = abs\_dist + |Bv\_e[i] - Bv[i]|$;
10:             $dist = dist + (Bv\_e[i] - Bv[i])$;
11:             If $abs\_dist = dist$
12:                 Compute the storage space $W$ of sub-flow table $c$;
13:                 $chose[num] = W$;
14:             EndIf
15:         EndFor
16:     EndFor
17:     Obtain the sub-flow table $c$ with the minimum storage space $W$ $(W > 0)$ from the
        vector $chose[]$;
18:     return sub-flow table $c$;
19: End
```

The H-SOFT is a heuristic algorithm invoking Algorithm 3 and Algorithm 4 for sub-flow tables splitting and merging operations to decrease the storage space of flow tables. Because the update of one flow entry will not affect the storage space of flow table significantly, and moreover, the frequent execution of the algorithm cause the remarkable consumption of CPU resources and memory space, the H-SOFT algorithm splits and merges the sub-flow tables if and only if the updating (i.e., adding and deleting) on the number of flow entries exceeds a pre-defined threshold. The experiment results shown in Section 5.2 have demonstrated that the H-SOFT algorithm significantly reduces the execution time while still maintaining the efficiency of storage space optimisation.

In H-SOFT algorithm, the execution of $SplitCluster()$ function in Algorithm 3 needs to traverse the entire k sub-flow tables to determine the flow table for splitting and calculate the storage space wastage for the maximum d number of fields, i.e., the complexity of the Algorithm 3 is $O(d^2N)$, where N denotes the number of flow entries for all flow tables. In contrast, the execution of $SplitCluster()$ function in Algorithm 4 do not need to traverse the $(N' + N'')$ flow entries, but determines the sub-flow tables c' and c'' to be merged by the structure of match field only. Note that N' and N'' denote the number of flow entries in sub-flow tables c' and c''. Thus, the time complexity of the Algorithm 4 is $O(k^2)$. Because $k < d$, therefore, the complexity of H-SOFT algorithm is $O(d^2N)$.

5. Simulation experiments and analysis

A software simulator based on Python programming language is developed to validate and evaluate the effectiveness of the proposed H-SOFT algorithm. All simulation experiments are collected from the server machine with Ubuntu 12.04.1 operating system, Xeon x5650 2.67GHz CPU and 24GB memory.

OpenFlow is currently in the process of development and seldom data of flow table is reported publicly. Large service providers, such as Google and Facebook, have adopted OpenFlow technology to deploy their data centres to improve the network performance and facilitate the flow control. However, it is difficult to obtain these data of flow table, and thus it is infeasible to use real data of flow table in OpenFlow networks to validate the effectiveness of the proposed H-SOFT algorithm.

To obtain the data of flow table for validation purpose, in this paper, we convert the header information of network packets into the corresponding entries of flow table. The data set of network packets is collected in 1000:1 samples from the real network traffic in the egress router of Guangzhou branch of China Science and Technology Network (CSTNET) [20] in one day, whose backbone network has covered 13 provinces including Beijing, Guangzhou, Shanghai, Kunming, etc. The initial entries and the new entries to be added into the flow table are generated with 19 fields: Time, Duration, Proto, IPv4_SRC, IPv4_DST, IPv6_SRC, IPv6_DST, TCP_SRC, TCP_DST, UDP_SRC, UDP_DST, ICMPv4_TYPE, ICMPv4_CODE, ICMPv6_TYPE, ICMPv6_CODE, TCP_FLAGS, TOS, PPS, and BPS, where *Time* and *Duration* fields denote the start time and duration of data flow, *PPS* and *BPS* fields are the packet sending rate and the number of bytes to send per second, and the other fields are commonly used in the traditional network. These fields have a similar structure of the field used in the OpenFlow flow table. The use of this data set for validation can reflect the effectiveness of H-SOFT algorithm on storage space compression for flow table.

In experimental results, the parameter p represents the compression rate of storage space for flow table. The larger of p obtains, the more effective of the algorithm performs. Parameter N and n denote the number of initial entries and the threshold upon which the algorithm starts to tune the structure of flow table, respectively.

5.1 The validation of H-SOFT algorithm

The H-SOFT algorithm assigns the entries into different sub-flow tables and each flow table has different valid fields. Fig. 1 depicts the value of p against the number of sub-flow tables, k, with initial flow table sizes N set to be 1000, 3000 and 5000, the threshold n set to be 200, and the number of updates on flow entries set to be 2000. From Fig. 1, we can find that with the increasing number of sub-flow tables k, the compression rate of storage space for flow table p increases, which is stable at ~72% finally. Thus, increasing the number of sub-flow tables can significantly reduce the storage space. By theoretically analysing the field structure of each entry in the flow table, the value of p should be around 72%~73%, which validates the accuracy of the proposed algorithm. After k reaches a certain value, the storage space of flow table does not have significant reduction. The results emphasise that obtaining the reasonable number of sub-flow tables for real deployment can reduce the expense and still maintain the performance of the algorithm.

5.2 The comparison between H-SOFT and optimal solution

The optimal solution for storage space compression is that all flow entries in all sub-flow tables need to be considered for adjustment due to the splitting or merging operation. However, the H-SOFT algorithm is a heuristic solution that takes into account the cases that only the flow entries involved in the flow tables for splitting and merging are considered for assignment within those flow tables.

To analyse the merit of the proposed heuristic algorithm, Fig. 2 and Table 2 depict the comparison on the execution time and compression rate of H-SOFT algorithm and optimal solution. In this section, the initial size of flow table varies from 1000 to 5000, the tuning threshold is set to be 200, the maximum number of sub-flow tables is set to be 10, and the number of new entries to be added into flow tables is set to be 2000. From Fig. 2, we can find that with the increasing size of the flow table, the run time required by the H-SOFT algorithm is increasing much smoother than that of optimal solution. However, the H-SOFT algorithm still maintains the compression rate for storage space of flow table in comparison with that obtained from optimal solution, as shown in Table 2.

5.3 The effect of threshold *n* on the performance of H-SOFT algorithm

To evaluate the effect of threshold n, upon which the algorithm starts to tune the structure of flow table, on the performance of the proposed H-SOFT algorithm, Fig. 3 depicts the compression rate p against the number of new entries to be added (i.e., 0, 1000, 5000, 10000, 20000, 30000 and 40000). The number of initial flow entries, N, is set to be 3000, the number of sub-flow tables, k, varies from 3 to 10, and tuning threshold, n, is set to be 200, 2000, 5000 and 10000.

From Fig. 3(a) under the case of 3 sub-flow tables, i.e., $k = 3$, we can find that the threshold, n, has significant impact on the value of p for 5000 and 10000 new entries. In particular, the smaller of n making a more frequent structure tuning of sub-flow tables by Algorithms 3 and 4, causes the higher compression rate, i.e., lower storage space. However, for the 1000 new entries, the current setting of threshold, n, does not have too much impact on the value of p, except for the case of $n = 200$, because the H-SOFT algorithm has not begun to adjust the structure of sub-flow tables. The increase in the number of new entries to be added, say 20000~40000, weakens the impact of threshold, n, on the performance of the proposed H-SOFT algorithm. That is because the sub-flow tables experience an increasing number of adjustment on their structures, resulting in the continuous decrease in the storage space. Another phenomenon can be found that the value of p has slight drop for some cases, such as 1000 and 5000 new entries with $n = 10000$ and 10000 and 20000 update entries with $n = 200$, because these new entries are added by the algorithm into the corresponding sub-flow tables directly, but do not cause the structure adjustment.

By comparing the Fig. 3(a)~3(d), we can find that increasing the number of sub-flow tables, k, decreases the impact of threshold, n, on the performance of the H-SOFT algorithm. That is because the a large number of sub-flow tables results in the more optimised storage space at the initial sub-flow tables created by Algorithm 2, and continuous adjustment of structure of sub-flow tables will get a slight impact on the performance of the algorithm.

The analysis emphasises the important findings that, in the actual deployment of the

H-SOFT algorithm, the threshold, n, can be set at a relative large value to decrease the usage for the resources of devices, while still remains the optimised storage space compression rate of flow table.

6 Conclusions

This paper has analysed the structure of match field in a flow table and proposed a Heuristic and efficient Storage space Optimization algorithm for Flow Table (H-SOFT) in an OpenFlow switch, which degraded the complex fields in match field of a flow table into multiple flow tables with simple fields based on the co-existence and conflict relationships among fields. In addition, the modelling of minimisation on the storage space of flow table has been given. Extensive simulation experiments have demonstrated that the H-SOFT algorithm can effectively reduce the storage space of flow table, and the number of sub-flow tables has significant impact on the compression rate of H-SOFT. In particular, when the number of sub-flow tables achieves a certain threshold, the compression rate of storage space cannot continuously increase. Moreover, with frequent updates on flow entries, the storage space compression rate of the flow table is stable and can achieve at ~70% with the complexity of the algorithm being $O(d^2N)$ where N is the number of flow entries and d is the number of fields. Furthermore, in comparison with the optimal solution, the H-SOFT algorithm can achieve the similar compression rate with much lower execution time.

Acknowledgement

This work is partially supported by the National Natural Science Foundation of China under Grant No. 61303241, National Program on Key Basic Research Project (973 Program) under Grant No. 2012CB315803, the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06010306, and the National High-Tech R&D Program of China (863 Program) under Grant No. 2013AA013501.

Reference

[1] Vaughan SJ. OpenFlow: The Next Generation of the Network?. *Computer* 2011; 44 (8): 13-15.

[2] Software-Defined Networking: The New Norm for Networks. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf [13 April 2012]

[3] Mckeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 2008; 38 (2): 69-74. DOI: 10.1145/1355734.1355746.

[4] OpenFlow 1.3.0 specification. https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf [25 June 2012].

[5] Jacobson V, Smetters D K, Thornton J D, Plass M F, Briggs N H, Braynard R L. Networking named content. *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, 2009. Communicaitons of ACM, January 2012; 55 (1): 117-124. DOI: 10.1145/2063176.2063204.

[6] Cao J, Chen B. memory-optimized RFC Packet Classification Algorithm Merge RFC. *Journal of Chinese Computer Systems* 2012; 33 (4): 865-868.

[7] Wooguil P, Sae-Woong B. FRFC: Fast Table Building Algorithm for Recursive Flow Classification. *Communications Letters IEEE* 2010; 14 (12): 1182-1184. DOI: 10.1109/LCOMM.2010.100810.100572.

[8] Lim H, Lee S, Swartzlander Jr E E. A new hierarchical packet classification algorithm. *Computer Networks* 2012; 56 (13): 3010-3022.

[9] Chang Y K, Chen H C. Layered Cutting Scheme for Packet Classification. *Proceedings of 2011 IEEE International Conference on Advanced Information Networking and Applications (AINA),* March 2011; 675-681. DOI: 10.1109/AINA.2011.70.

[10] Jiang W, Prasanna V K. Energy-efficient multi-pipeline architecture for terabit packet classification. *Proceedings of IEEE Global Telecommunications Conference,* 2009. IEEE Communication Society, 2009; 1-6. DOI: 10.1109/GLOCOM.2009.5426226.

[11] Cheung G, McCanne S. Optimal routing table design for IP address lookups under memory constraints. *Proceedings of Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, 1999. 3: 1437-1444. DOI: 10.1109/INFCOM.1999.752164.

[12] Yang T, Yuan B, Zhang S, et al. Approaching optimal compression with fast update for large scale routing tables. *Proceedings of 20th International Workshop on Quality of Service (IWQoS)*, 2012. 1-9. DOI: 10.1109/IWQoS.2012.6245978.

[13] Karpilovsky E, Caesar M, Rexford J, Shaikh, A, van der Merwe, J. Practical Network-Wide Compression of IP Routing Tables. *IEEE Transactions on Network and Service Management* 2012, 9 (4): 446-458. DOI: 10.1109/TNSM.2012.081012.120246.

[14] Zeng KY, Yang JH. Towards the optimization of access control list. *Journal of Software*, 2007; 18 (4): 978−986.

[15] Daly J, Liu A X, Torng E. A Difference Resolution Approach to Compressing Access Control Lists. *Proceedings of International Conference on Computer Communications (INFOCOM)*, 2013; 2040-2048. DOI: 10.1109/INFCOM.2013.6567005.

[16] Liu A X, Torng E, Meiners C R. Compressing network access control lists. *IEEE Transactions on Parallel and Distributed Systems*, 2011. 22 (12): 1969-1977. DOI: 10.1109/TPDS.2011.114.

[17] Curtis AR, Mogul JC, Tourrilhes J, Yalagandula P, Sharma P, Banerjee S. DevoFlow: Scaling flow management for high performance networks. *Proceedings of the Special Interest Group on*

Data Communication (SIGCOMM), 2011. Toronto: ACM Press, 2011; 254-265. DOI: 10.1145/2018436.2018466.

[18] Yu M, Rexford J, Freedman MJ, Wang J. Scalable flow-based networking with DIFANE. *Proceedings of the Special Interest Group on Data Communication (SIGCOMM),* 2010. New Delhi: ACM Press, 2010; 351-362. DOI: 10.1145/1851182.1851224.

[19] Matsumoto N, Hayashi M. LightFlow: Speeding up GPU-based flow switching and facilitating maintenance of flow table. *Proceedings of IEEE 13th International Conference on High Performance Switching and Routing (HPSR)*, 2012; 76-81. DOI: 10.1109/HPSR.2012.6260831.

[20] CSTNET: http://www.cstnet.net.cn/english/index.htm

Table 1. The fields in Match Field of Flow Table in OpenFlow1.3

Fields	Description
IN_PORT	Switch input port.
IN_PHY_PORT	Switch physical input port.
METADATA	Metadata passed between tables.
ETH_DST	Ethernet destination address.
ETH_SRC	Ethernet source address.
ETH_TYPE	Ethernet frame type.
VLAN_VID	VLAN ID.
VLAN_PCP	VLAN priority.
IP_DSCP	IP DSCP (6 bits in ToS field).
IP_ECN	IP ECN (2 bits in ToS field).
IP_PROTO	IP protocol.
IPV4_SRC	IPv4 source address.
IPV4_DST	IPv4 destination address.
TCP_SRC	TCP source port.
TCP_DST	TCP destination port.
UDP_SRC	UDP source port.
SCTP_SRC	UDP destination port.
UDP_DST	SCTP source port.
SCTP_DST	SCTP destination port.
ICMPV4_TYPE	ICMP type.
ICMPV4_CODE	ICMP code.
ARP_OP	ARP opcode.
ARP_SPA	ARP source IPv4 address.
ARP_TPA	ARP target IPv4 address.
ARP_SHA	ARP source hardware address.
ARP_THA	ARP target hardware address.
IPV6_SRC	IPv6 source address.
IPV6_DST	IPv6 destination address.
IPV6_FLABEL	IPv6 Flow Label
ICMPV6_TYPE	ICMPv6 type.
ICMPV6_CODE	ICMPv6 code.
IPV6_ND_TARGET	Target address for ND.
IPV6_ND_SLL	Source link-layer for ND.
IPV6_ND_TLL	Target link-layer for ND.
MPLS_LABEL	MPLS label.
MPLS_TC	MPLS TC.
MPLS_BOS	MPLS BoS bit.
PBB_ISID	PBB I-SID.
TUNNEL_ID	Logical Port Metadata.
IPV6_EXTHDR	IPv6 Extension Header pseudo-field

Table 2. The comparison on compression rate of flow table between the H-SOFT and optimal solution

Initial size of flow table		1000	2000	3000	4000	5000
Number of entries after updating		1289	1766	2558	3578	4695
p	H-SOFT	0.71694	0.71319	0.70379	0.70268	0.69696
	Optimal solution	0.71326	0.71752	0.71689	0.71766	0.72021

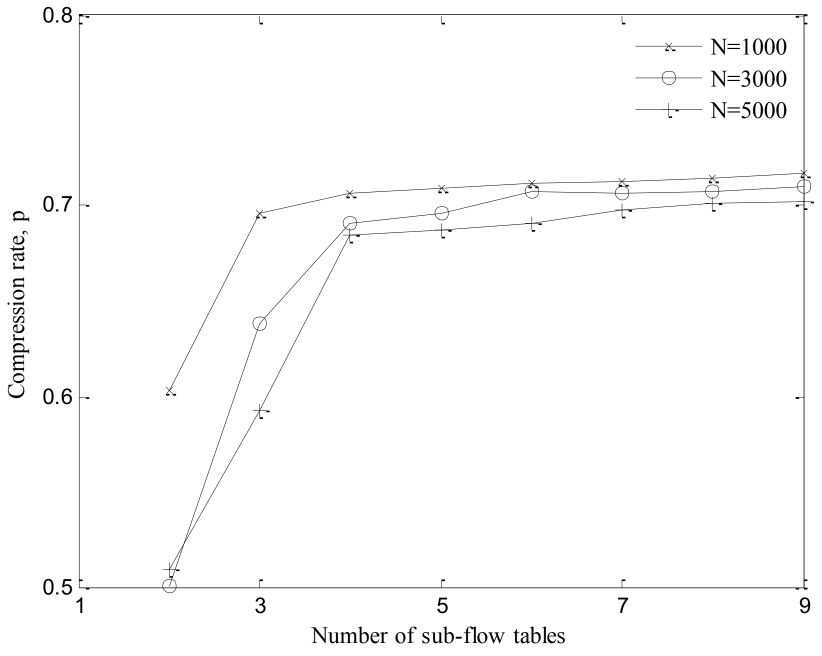

Fig. 1. The optimisation for the storage space of flow table by H-SOFT algorithm with the increasing number of sub-flow tables

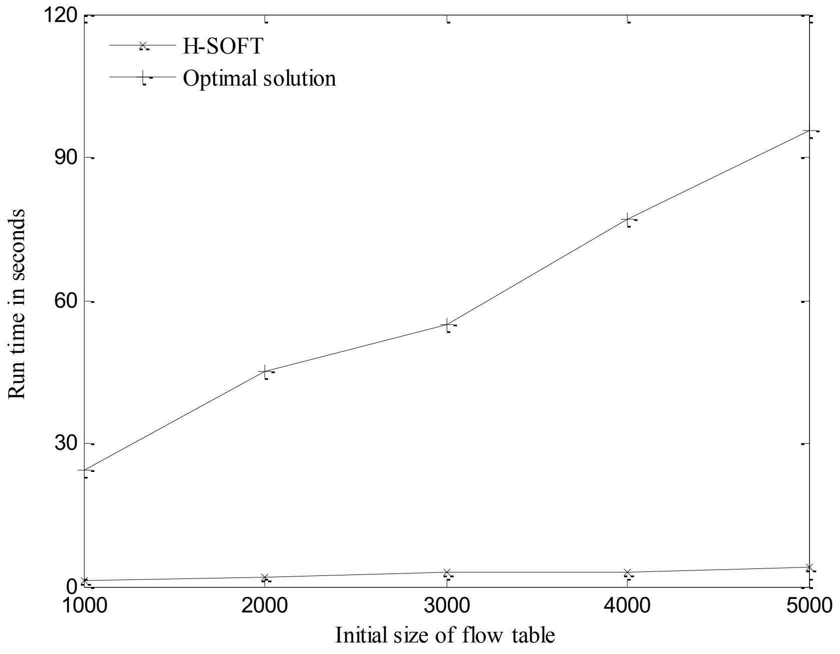

Fig. 2. The comparison of run time between H-SOFT and optimal solution

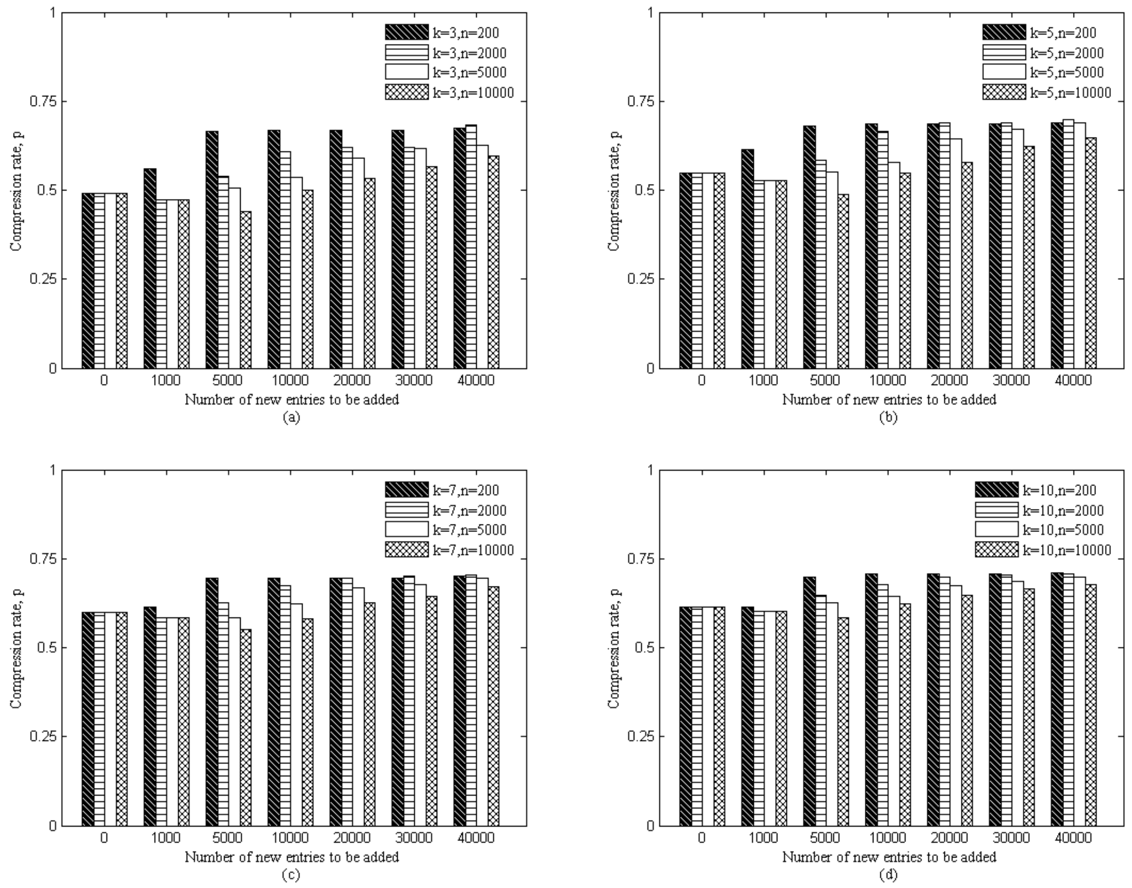

Fig. 3. The effect of threshold *n* on the performance of H-SOFT algorithm