Range Encoding and Hash table based Packet Classification for Global View Networking

Yeim-Kuan Chang, Yi-Hao Lai, and Che-Wei Li
Department of Computer Science and Information Engineering
National Cheng Kung University, Taiwan

***Abstract*—Packet classification is an important functionality of the Internet router for many network applications. With the emergence of software-defined networking (SDN), packet classification for global view networking is used to search the actions taken at multiple routers, not only at a single router. The control plane provides a global view of the network, which allows applications to identify the network-wide behavior of a packet, defined as the combination of actions taken at all routers. In this paper, we propose a two-layer scheme named range encoding hash table (REHT) that can search the network-wide behaviors of packets efficiently. In layer one, the header field values of all fields are encoded separately. In layer two, hash tables are used for the encoded values to achieve high classification speed. Based on our experiments using real network configurations, REHT performs much faster than BDDs and MDD schemes.**

Keywords- Packet classification; IP lookup; Encoding; Hash table

I. INTRODUCTION

A router forwards packets between networks. When a packet comes in, the router uses packet headers to determine the next hop obtained from routing table. Also, routers need packet classification [2] to support many network applications by classifying packets into *flows*. Flows are specified by *classifier*. The packets classified as a flow are processed in the same manner as defined in the action associated with the flow. Each rule specifies a flow based on five header fields, source and destination address IP fields, source and destination port fields, and protocol field. Each field value may be formatted as a prefix, a range, or a singleton value. Each rule also has a priority. When a packet matches multiple rules, this packet is classified as the flow with the highest priority.

With the development of Internet and emergence of SDN, packet classification is no longer just to classify the packets at a single router. The network behaviors for multiple routers need to be considered at the same time. SDN architecture decouples network control and forwarding function. OpenFlow is the de facto standard for SDN where the control plane operated as a centralized controller provides a global view of the network to allow applications to identify network-wide behaviors of packets. Network-wide behaviors are defined by the routing tables and rulesets in all routers of the network. Network-wide behavior can display how a packet traverses in the network and whether a packet will be discarded. Figure 1 shows the global view of a network with 3 routers. Each router maintains a routing table and one rule table. Figure 1(h) shows the routing behavior table computed from three routing tables. Figure 1(g) shows the network topology. The 2-D header space of these rule tables consists of six disjoint blocks for four distinct rule behaviors in Figure 1(j). Each rule behavior is represented by a three-action tuple

	Dst IP	next port
R1	2:3	A1
R2	4:5	A2
R3	0:3	A4
R4	6:6	A3
R5	*	drop

(a) Routing table A

	Dst IP	next port
R1	2:3	B1
R2	6:7	B2
R3	5:5	B3
R4	4:7	B4
R5	*	drop

(b) Routing table B

	Dst IP	next port
R1	0:1	C3
R2	6:7	C2
R3	2:3	C1
R4	*	drop

(c) Routing table C

Field1	Field 2	Action
*	0:3	permit
*	6:7	permit
2:3	*	permit
*	*	deny

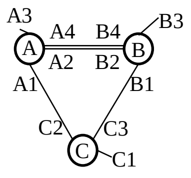
(d) Rule table A

Field1	Field2	Action
*	0:3	permit
*	4:5	permit
0:3	*	permit
*	*	deny

(e) Rule table B

Field1	Field2	Action
*	4:7	permit
1:1	2:3	permit
*	*	deny

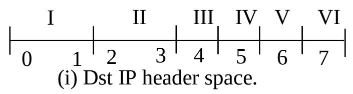
(f) Rule table C



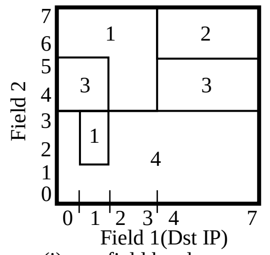
(g) Network topology

	A	B	C
I	A4	drop	C3
II	A1	B1	C1
III	A2	B4	drop
IV	A2	B3	drop
V	A3	B2	C2
VI	drop	B2	C2

(h) Routing behaviors



(i) Dst IP header space.

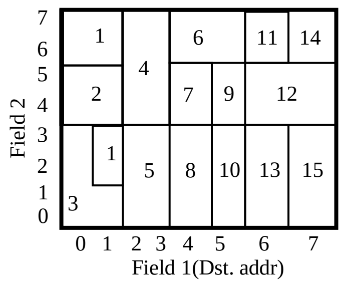


(j) two-field header space

	A	B	C
1	P	P	P
2	P	D	P
3	D	P	P
4	P	P	D

(k) Rule behaviors

Figure 1. Example of a global view network.



1	(A4, Drop,C3)
2	(Drop, Drop,C3)
3	(A4, Drop, Drop)
4	(A1,B1,C1)
5	(A1,B1, Drop)
6	(A2, Drop, Drop)
7	(Drop,B4, Drop)
8	(A2,B4 Drop)
9	(Drop,B3, Drop)
10	(A2,B3, Drop)
11	(A3, Drop,C2)
12	(Drop,B2,C2)
13	(A3,B2, Drop)
14	(Drop, Drop,C2)
15	(Drop,B2, Drop)

Figure 2. Example of network-wide behaviors.

(action$_A$, action$_B$, action$_C$) in Figure 1(k). After obtaining the routing behavior and rule behavior, we obtain the network-wide behavior of the packets as shown in Figure 2.

Our scheme allows SDN applications running on the control plane to identify network-wide behaviors. If a packet cannot reach its destination, then the border router can directly drop it. The main task is to classify the incoming packets with the classifiers of all routers. The methods designed for a single router is not efficient for global networks since we have to repeat the process until we get all flows. To classify a network-wide behavior efficiently, we need to combine all classifiers into one. However, conventional methods like HiCuts [4] and HyperCuts [7], do not support network-wide behaviors since the search space becomes too complicated.

In this paper, we propose a novel scheme named *Range Encoding Hash Table (REHT)*. REHT is a two-layer hash table-based scheme for solving packet classification problem for global view networking. In layer one, we construct five range encoders for five fields by classifiers. In layer two, we use hash tables to record rules. For an incoming packet, we encode its five field values and access hash tables to obtain its network-wide behavior.

The rest of this paper is organized as follows. Section II briefly reviews related work. Section III shows the overview and details of the proposed REHT scheme. Also, optimization is proposed for ACL rule table. Grouping optimizations which can be adopted by REHT to improve memory usage are proposed in Section IV. The performance evaluation is shown in Section V and conclusions are shown in the last Section.

II. RELATED WORK

In past years, numerous packet classification schemes have been proposed in the literature. These schemes can be categorized as software solutions and hardware solutions. Software algorithms can also be divided into two categories, decision-tree and field decomposition schemes. HiCuts [4], HyperCuts [7], and EffiCuts [1] are well-known decision trees. BDDs [10] and MDD [11], which can solve network-wide behavior problem are also decision trees. Decision trees see the packet classification problem in the geometric view to cut the search space into smaller subspaces. Field decomposition schemes include BV [5] and RFC [6]. They perform independent searches on each field and combine the intermediate results of all fields to obtain the final results. Hardware schemes usually use parallel search engine, such as Ternary content addressable memory (TCAM) and pipelined design using FPGA. Other encoding schemes can be found in [2][12][13].

Binary decision diagram [8] (BDD) is a decision tree that is used to represent a Boolean function. BDDs can be considered as a compressed representation of sets or relations. In [10], they proposed a control plane tool for packet behavior identification, as known as network-wide behaviors. They first convert forwarding table and rule table to a list of *predicates*. A predicate represents a discontinuous space for an output port or action. Then, they compute separate sets of *atomic predicates* for rule and forwarding predicates. An atomic predicate can be represented by a BDD. For *n* atomic predicates, it can be represented by *n* BDDs. Then, they build the AP tree based on atomic predicates to reduce the number of times to search BDDs.

Boolean functions can be merged into a single multi-valued function to represent a whole classifier by itself. This multi-valued function is represented by a data structure named multi-value decision diagram [9] (MDD), a variant of BDD. In [11], they proposed some optimized method for MDD to update a new behavior. Also, they proposed an algorithm to accelerate classification process by regarding a bunch of header bits as a single variable to analyze time-space tradeoff.

III. PROPOSED SCHEME

The packet classification (PC) for global view networking depends on multiple routing and rule tables. The address space cutting procedure to get an action, or a network-wide behavior is complicated. We can divide the PC problem of global view network into problems of identifying the routing behavior and rule behavior for the incoming packets. Then we can get the final network-wide behavior by combining the identified routing and rule behaviors. A routing behavior refers to the set of next ports defined by all routing tables of the routers. Similarly, a rule behavior refers to the set of actions defined by all rule tables of the routers.

After obtaining routing and rule behaviors of a packet, we have to combine the next ports and actions. If the rule action of a router is permit, its next port of network-wide behavior remains the same. Otherwise, it changes with rule action. Assume the routing and rule behaviors of a packet with header values (6, 2) are (A3, B2, C2) and (permit, permit, deny), respectively, as shown in Figure 1. After combining operations, the output port of router C becomes “drop”.

The proposed Range Encoding Hash Table (REHT) is a two-layer field-independent based scheme that uses hash tables to store the rule tables. For an incoming packet, REHT encodes the five field values of the packet separately and computes the routing behavior based on the destination IP field encoded value in layer one. In layer two, REHT uses the encoded values of layer one to access the hash tables and compute rule behaviors. Finally, we can obtain the network-wide behavior by combining the routing and rule behaviors.

We first combine all routing tables into a big integrated routing table and all rule tables into a big integrated rule table. Then, we divide the address space of each field into several intervals based on the concept of elementary interval. For the destination address field, we construct the destination address elementary intervals by the destination IP field values of both routing table and rule table. Then, we encode field values into interval numbers call *interval ID*. Layer one contains five range encoders where each range encoder inputs the corresponding field values and outputs the interval IDs. We call the encoded value of the source address field as *source address interval ID*, the encoded value of the destination address field as *destination address interval ID*, and so on. For routing behaviors, we use the integrated routing table to precompute the routing behavior where each destination address interval ID corresponding to a routing behavior.

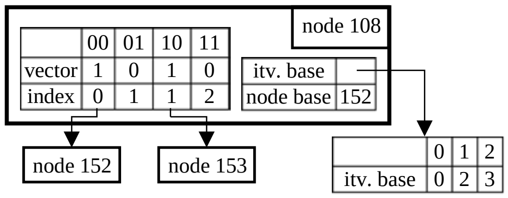


Figure 3. Example node of 2-bit multiway range tree.

Based on encoded values of five dimensions, the rule behavior can be easily stored in the hash tables and the queries of rule behaviors from incoming packet headers can also be computed efficiently.

We know that each field value in five-dimension rule set is either exact value or wildcard (i.e., don't care). Since we cannot hash a rule containing a wildcard without serious duplications, we simple divide the rules into groups based on whether the value of a field is exact value or wildcard, and each group is given a hash table. For each group, we can only consider the field values that are exact values. After accessing all the hash tables from incoming packet headers, we can obtain rules behaviors by the priority encoder. Since we need to access all hash tables to get the rule behavior and it takes a lot of memory accesses, we use possibility bitmap to reduce unneeded memory accesses, described in detail later.

A. *Range Encoder*

We first build the respective elementary intervals [3] for each field by both integrated routing and rule tables. The purpose of the encoder is to get the interval ID to which the field value belongs. As the size of each field is different, we use three different methods for the encoder to balance number of memory accesses and memory usage as follows.

1) *Direct entry mapping for port field (small size)*

For the field of length *m*, we use an array of 2^m entries to directly map the address to the corresponding interval ID with only one memory access.

2) *Multiway range tree for IP address (large size)*

Multiway range tree is similar to multiway tries, except for building by endpoint not prefix. Multiway range tree consists of internal nodes and leaf nodes. For the *k*-bit multiway range tree, each internal node holds an array with 2^k element corresponding to the value of the *k*-bit chunk in the input key and each element contains a vector and an index. Besides, it holds a node base and an array pointer which points to an array with *n* interval base where *n* depends on the max value of index. The vector is configured so that bit-1 indicates there is a descendant node where the node ID is the sum of base and corresponding index, and the bit-0 indicates there is no descendant node. Also, each leaf node holds an interval base array with 2^k elements. Figure 3 shows an example of the data structure of 2-bit multiway range tree.

The searching process starts at the root node and the interval ID is set to 0. Each traverse to a node, we use the most-significant *k*-bit of the input address as the searching key, and then we add the i^{th} item of the interval base array to interval ID which *i* is the index corresponding to the searching key. If the corresponding vector is 1, we shift the input address *k* bits to the left and go to the descendant node. Otherwise, we output the interval ID and terminate the process.

	Router/Action	F1 itv ID	F2 itv ID	F3 itv ID	Group	key	Hash
R1	B/permit	0(000)	0(000)	6(110)	1	000000110	5
R2	A/permit	3(011)	5(101)	2(010)	7	011101010	13

Hash table 1		
Idx	Key	Behavior
5	**000000110**	D**P**D
6	000000000	DDD

Hash table 7			
Idx	Key	Behavior	
13	**011101010**	**P**DD	
14	000000000	DDD	

Possibility bitmap			
Itv	field 1	field 2	field 3
2	0**1**000000	0**1**000000	00000000
3	0**1**000000	0**1**000000	00000000
4	0**1**000000	0**1**000000	00000000
5	0**1**000000	0**1**000000	00000000
6	0**1**000000	0**1**000000	0**1**000000

After inserting R1.

Possibility bitmap			
Itv	field 1	field 2	field 3
2	01000000	01000000	0000000**1**
3	0100000**1**	01000000	00000000
4	01000000	01000000	00000000
5	01000000	0100000**1**	00000000
6	01000000	01000000	01000000

After inserting R2.

Figure 4. Example of inserting a rule.

3) *Condition check for protocol (few distinct values)*

For the field with few dissimilar values, such as the protocol field which only consists of TCP, UDP and don't care, we can use condition check to get the interval ID. Specifically, there are only three interval IDs that corresponds to TCP, UDP, and others. So, we can easily to obtain the interval ID for a protocol number by simply using if-else condition check.

B. *Hash Table*

Given *n*-field rules, we divide them into 2^n groups with *n*-bit group IDs. The rule whose field-*i* value is wildcard must belong to the group whose bit-*i* is 0. Each group is given a hash table. Each entry of hash tables holds a key initialized to 0 and a rule behavior initialized to default. To distinguish the rule behaviors of incoming packets and the rule behaviors recorded in hash tables, we call the rule behaviors recorded in hash tables *group behaviors*. The packets' rule behaviors are determined by the query results of group behaviors from incoming packet headers in each hash table.

1) *Insert Rules*

To insert a rule into hash table, we first convert every field value of the rule into interval IDs and combine them as the inserting key. Given hash function *h*, the corresponding index of this rule is *h*(inserting key). If the key of this entry is 0 or equal to the inserting key, we set the value of key to inserting key and update the group behavior as shown in Figure 4. If the key of this entry is neither 0 nor inserting key, which means a collision, we increase the number of entries for hash table until we can put all the rules into hash table without collision.

If any field value of a rule covers multiple intervals, we need to duplicate this rule *m* times where *m* is the product of the number of intervals that each field covers. It is an important issue that a rule may be duplicated hundreds of times in the worst case. To solve this problem, we use two optimized grouping schemes which will be introduced later. In our experiment, we also use cuckoo hashing for layer two. It has better memory usage, but it needs more memory accesses since it has multiple hash functions.

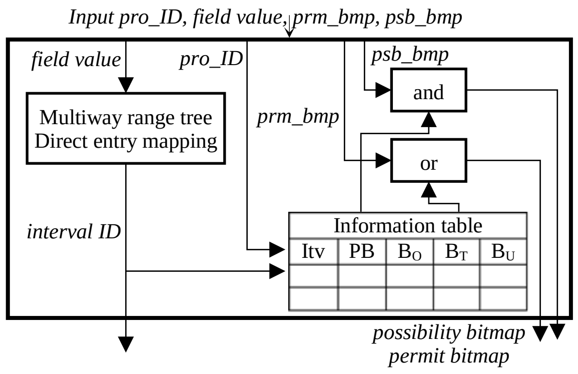


Figure 5. Data structure of the field encoder in layer one.

2) Possibility Bitmap

The possibility bitmap is used to confirm which hash tables an interval ID may hit. Given *n* hash tables, we configure a set of *n*-bit possibility bitmap for each interval ID, and each bit corresponds to a hash table. The most significant bit of the bitmap corresponds to hash table 0. If bit-*i* of possibility bitmap for an interval ID is 1, the hash key with this interval ID may hit hash table *i*. Otherwise, the hash key with this interval ID will never hit hash table *i*. Figure 4 shows how to set possibility bitmaps when inserting two rules R1 and R2, where non-relevant entries are not shown.

3) Searching Process

To find out the rule behavior of an incoming packet, we use the interval IDs converted from the encoders of layer one as the searching key, and we use and operation on the possibility bitmaps of these intervals. Then, we use the searching key to access the hash tables which the corresponding bit of possibility bitmap is 1. To access a hash table, we identify the key in the corresponding index with searching key. If these two are identical, this group behavior is matched. Otherwise, this hash table is missed. Finally, we can obtain the rule behavior by combining all of matched group behavior from priority encoder.

C. Optimized Scheme for ACL Rule

We propose an optimized scheme for the packet classification with ACL rule, such as Stanford backbone network. Based on the property of ACL rule and network-wide behavior, we can improve the performance.

1) ACL Rule

As we known, ACL rule contains five fields and only two kinds of actions, permit and deny. In the optimized scheme, we replace the rules of deny action with the rules of permit action in the case of maintaining the original property. Since the actions of rules after replacement are all permit, for group behaviors, we can use *n* bits to represent *n* actions of *n* routers where bit 1 represents permit and bit 0 represent deny. Then, we can use or operation to obtain the rule behaviors instead of priority encoder.

Furthermore, the source port field of ACL rule is always wildcard, so we can use 2^4 hash tables to record the group behaviors. Also, based on the analysis of ACL rule, some groups may be empty. For ACL rule in Stanford backbone network, the number of hash tables is ten.

```
Input Header H, the input header;
Initial pmb = (all false), permit bitmap;
Initial pbb = (all true), possibility bitmap;
pro_ID = pro_enc(H.protocol);
dstIP_ID = dstIP_enc(H.dstIP, pro_ID, &pmb, &pbb);
srcIP_ID = srcIP_enc(H.srcIP, pro_ID, &pmb, &pbb);
dstPort_ID = dstPort_enc(H.dstPort, pro_ID, &pmb, &pbb);
rout_b = routing _behavior_table[dstIP_ID];
key = key_combiner(srcIP_ID, dstIP_ID, dstPort_ID, pro_ID);
HT = 1;
For i = 0 to 4
START
    If(pmb == all true) Return rout_b;
    If(pbb & HT) hash_table(i, key, &pmb);
    HT = HT << 1;
END
```

Figure 6. Searching process in optimized scheme.

We know that some hash tables only contain one exact field like source or destination address, which means we can obtain group behavior from this field interval ID instead of hash procedure. So, we can remove these two hash tables and put the rules of these tables into the source and destination address *information table* which use the same interval index as possibility bitmap. In other words, each interval *i* = 1 to S is associated with a possibility bitmap and a group behavior. Moreover, since the protocol field only contains three values, TCP, UDP, and don't care, we can merge the two hash tables of which exact fields and wildcard fields are the same except for the protocol field by using three group behaviors to record the case of TCP, UDP and don't care, respectively, denoted by (B_T, B_U, B_O). We then can reduce the number of hash tables to five.

2) Network-wide Behavior

We know that we can get the network-wide behavior by combining the routing behavior and rule behavior, but there are two cases that we can get the network-wide behavior without complete procedure. In the first case, if the routing behavior of an incoming packet will eventually drop the packet, then we do not have to get its exact rule behavior. Since no matter what its rule behavior is, the packet will drop. In the second case, since all the actions of rules are permit, for an incoming packet, if one of matched group behaviors for switch *k* is permit, the action of rule behavior for switch *k* is permit. If the rule behavior of an incoming packet is all permit, its network-wide behavior is its routing behavior. So, in the optimized scheme, we can access to the hash table one by one and use or operation to record the matched group behavior. Once all the actions of the rule behavior are permit, the query process can be terminated.

3) Permit Bitmap

In the searching process, we use a set of permit bitmap to record the current rule behavior of the packet. The length of permit bitmap is same as the rule behavior, and it is initialized to all false (not permit). Every time accessing to a

TABLE I. STATISTICS OF NETWORK CONFIGURATION.

	Internet2	Stanford
# of routers	9	16
# of prefixes (FIB)	126,017	757,170
# of rules (ACL)	0	1,584
# of header bits of interest	32	88

TABLE II. STATISTICS OF STANFORD ACL TABLE.

	Src. addr	Dst. addr	Dst. port	Protocol
# of intervals	418	226	55	3

Group	0	1	2	3	4	5	6	7
# of rules	0	0	0	8	16	35	0	57
Group	8	9	10	11	12	13	14	15
# of rules	77	31	0	23	161	11	0	29

	Information table	Hash table
# of rules	167	281

field encoder or a hash table, the permit bitmap is updated if a group behavior is matched. Once the permit bitmap is all true (permit), the query process can be terminated. If the permit bitmap is not all true at the end of query process, the rule behavior is represented by permit bitmap. Figure 5 shows the complete field encoder in layer one. Figure 6 shows the pseudo code of searching process in optimized scheme.

IV. GROUPING OPTIMIZATIONS

As described earlier, duplication is an important issue of packet classification that a rule may be duplicated hundreds of times in the worst case. To solve this problem, we propose another two grouping methods for hash tables. In our experiment, we can improve the performance by more than ten times. In other words, we can reduce the total number of duplications to less than one tenth.

The normal grouping is based on whether each field is exact field or wildcard field. However, in some rules, the length of the source and destination address field may be short, or the field value may cover many intervals. We call these heavy rules that duplicate many times. Our goal is to reduce this kind of rules. In the first grouping method, grouping by prefix length, we define the field with length less than or equal to k as wildcard. For example, given $k = 2$, the field value 128.0.0.0/1 is a wildcard field. In our experiment, this method can probably reduce the total number of duplications to less than half. In the second grouping method, grouping by the number of duplications, we decide whether a field value is wildcard or exact field according to the number of intervals it covers. Then, we construct respective elementary interval for wildcard field and exact field. For incoming packets, each encoder output two interval IDs, one used for wildcard field, and another used for exact field. In our experiment, the second grouping method can reduce the total number of duplications to less than one tenth.

V. PERFORMANCE EVALUATION

Our scheme is evaluated with two real networks: Internet2 and Stanford backbone networks [11]. The network statistics of Internet2 and Stanford backbone network are shown in Table I.

TABLE III. STATISTICS OF MULTIWAY RANGE TREE.

Header field		# of nodes	Memory (KB)
Internet2 Dst. address	8-8-8-8	18,416	898.5
	16-8-8	18,241	985.1
	12-10-10	11,849	1,974.5
Stanford Src. address	8-8-8-8	244	11.7
	16-8-8	230	66.3
	12-10-10	151	27.5
Stanford Dst. address	8-8-8-8	1,052	46.5
	16-8-8	1,033	108.7
	12-10-10	547	87.5

TABLE IV. MEMORY USAGE.

	Traditional hashing	Cuckoo hashing
Memory(KB)	767.33	255.77

	Routing	Src. addr	Dst. addr	Dst. port
Memory(KB)	5.44	2.71	1.46	0.36

TABLE V. RESUTLS OF THREE GROUPING.

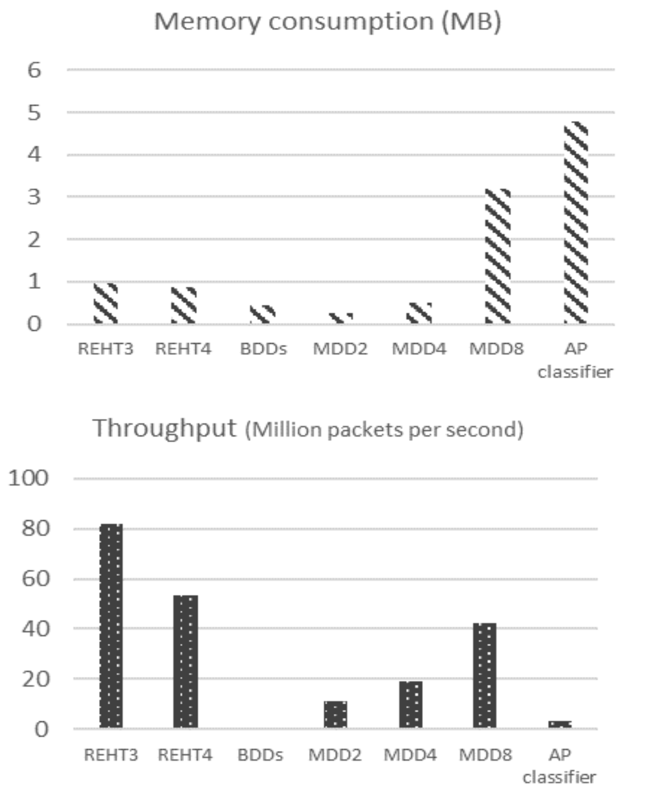
Grouping	# of items in hash tables
Original	17,034
Optimization 1	7,307
Optimization 2	1,092

A. *Experimental Analysis*

For our proposed scheme, the performance depends on an important factor, the number of intervals in each dimension. If the number of intervals is large, the data structure for encoder is large. Also, the duplications in hash tables may be large. Furthermore, the number of intervals in destination address field is larger than other fields, so the most important factor is the number of intervals in destination address field. The average number of intervals in Internet2 integrated routing table is 14383, and the number of routing behaviors is 457. The number of intervals in Stanford integrate routing table is 2086, and the number of routing behaviors is 507. As a result, the multiway range tree for Stanford is better than Internet2. Table II shows the statistic of Stanford integrate ACL rule table. In optimized scheme, there are probably 37% of the rules that can be recorded in the information tables.

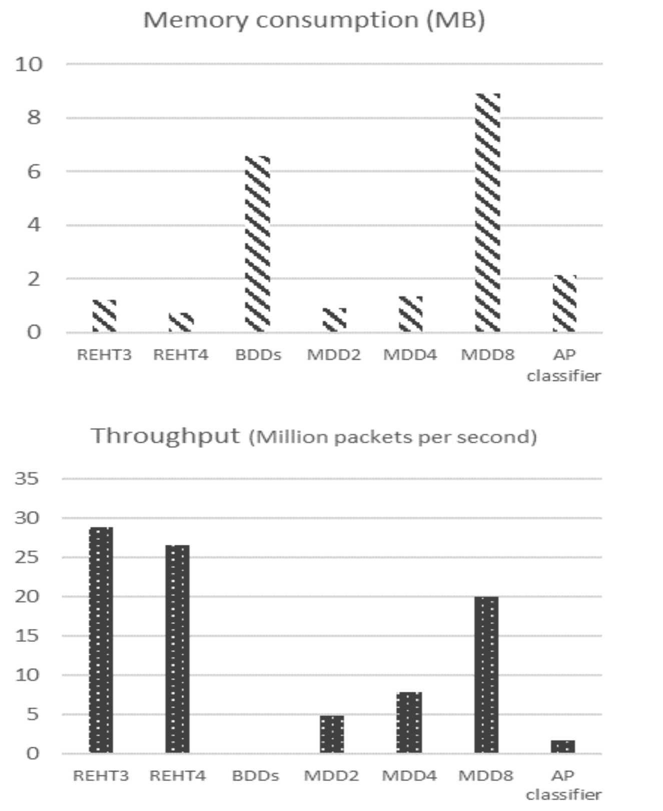
B. *Experimental Results*

We show the performance results of the proposed scheme in two parts, range encoding and hash procedure. In range encoding, the multiway range trees for the IP address fields are implemented in three different configurations, denoted by 8-8-8-8, 16-8-8, 12-10-10. Notation 8-8-8-8 means that the multiway range tree is organized as a four-level data structure and each level takes 8 bits of the 32-bit address space. Notations 16-8-8 and 12-10-10 mean that the multiway range tree is organized as a three-level data structure such that the first level takes 16 and 12 bits; the next two levels take 8 and 10 bits each, respectively. Table III shows the statistics of multiway range tree. The number of nodes is associcated with the number of intervals. By comparing three-level and four-level data structure, three-level needs more memory consumption but less memory accesses. By comparing



	REHT3	REHT4	BDDs	MDD2	MDD4	MDD8	AP classifier
Memory	0.962	0.877	0.454	0.26	0.51	3.2	4.79
Throughput	82.05	53.33	0.085	11.3	18.95	42.6	3.4

Figure 7. Performance comparison of Internet2.



	REHT3	REHT4	BDDs	MDD2	MDD4	MDD8	AP classifier
Memory	1.23	0.753	6.568	0.9	1.35	8.89	2.15
Throughput	28.828	26.6	0.006	4.95	7.86	20.02	1.8

Figure 8. Performance comparison of Stanford.

configurations 16-8-8 and 12-10-10, configuration 16-8-8 is more suitable for Internet2 and configuration 12-10-10 is more suitable for Stanford backbone network.

Hash procedure is implemented by two hashing methods, traditional hashing and cuckoo hashing. Traditional hashing in REHT is implemented as follows. We set the size of the hash table as (3 * *# of hash items*) and each hash entry can hold three items. According to our experiment, it is the smallest hash table size that can record all rules without collision. Table IV shows the memory consumption of two hashing method for Stanford. The memory consumption of cuckoo hashing is smaller, but traditional hashing needs less memory accesses.

Table IV also shows the memory consumption of information table. Routing information table contains the routing behaviors. Other information tables contain the possibility bitmaps and group behaviors. TABLE V shows the results of three grouping method. Optimization 1 is divide the rules by field length, and optimization 2 is divide the rules by number of intervals that a field value covers. Optimization 2 can reduce the number of items to less than 10%.

We compare our proposed scheme with BDDs and MDD schemes by the same network configurations, Internet2 and Stanford backbone networks. We use instruction 'rdstc' (read time stamp counter) to measure the CPU clock ticks of the searching process and compute the average throughput that is defined as CPU cycles per search. The performance results are shown in Figure 7 and Figure 8. REHT3 is constructed by range encoding configuration 16-8-8 (Internet2) or 12-10-10 (Stanford), and REHT4 is constructed by range encoding configuration 8-8-8-8. Both REHT3 and REHT4 use traditional hashing since it has better classification speed. MDD2/4/8 represents 2/4/8-bit multiway MDD. Since the REHT3 and REHT4 for one dimensional lookup just consist of multiway range tree, the number of memory accesses of REHT3/4 is equal to or less than 3/4. The number of memory accesses of MDD2/4/8 for Internet2 is 16/8/4. So, the throughput of REHT is much better than BDDs and MDD (k = 2/4). For Stanford backbone network, REHT3 has the highest throughput and REHT4 has the smallest memory usage. REHT4 also has second high throughput. The worst case number of memory accesses of REHT for Stanford is sum of the accesses in range encoders and hash tables. The worst case of REHT3 is 12, and the worst case of REHT4 is 14. The number of memory accesses of MDD8 for Stanford is always 11. However, REHT can avoid unnecessary memory accesses, so the throughput of REHT is better than MDD8. For five-dimension header, the memory usage of decision-tree based schemes increase. On the other hand, the memory usage of REHT do not increase too much. Also, REHT can reach fast classification speed due to hash procedure. Since AP classifier [10] optimized BDD by reducing the number of searched BDDs, its throughput is only better than original BDD and its memory usage is higher than original BDD.

For throughput, as our single operation is simple enough like BDDs and MDD, the number of memory accesses of REHT is less than or equal to BDDs and MDD. This is why the throughput of the proposed scheme is better than BDDs and MDD. For memory usage, as described in the grouping optimization, duplication is an important issue in our encoding scheme. It may cause serious rule duplication when the IP/port fields have more wildcard (but actually not). For the limitation of REHT, the rule table configuration can't consist of a deny rule that has a higher priority than any permit rule since we only consider the permit action in our method. It can be extended to the configuration with more than one action, which is our future work.

To allow the proposed scheme working with switches, we only need to find another way to efficiently encode the mac address which is a singleton value field. Also, we have add another field VLAN ID in the rules to make sure to which VLAN the classified network behaviors are related. Classbench is a suite of tools for benchmarking packet classification algorithms to produce synthetic filter or rule sets that accurately model the characteristics of various types of networks. We need such tool to support that REHT is suitable for some kinds of network configuration. For the future work, we try to develop a rule generator for different network configurations. The difference from ClassBench is that the rule generator for global view networking has to identify the correctness for packet routing. Also, the rules gererated for different routers should have some common prefixes. Other than generating the tables to model the characteristics of real networks, we have to do research to identify that every route and rule action in the global network is reasonable.

VI. Conclusions

In this paper, we proposed the Range Encoding Hash Table (REHT) packet classification scheme for global view networking. For multiple routing tables and rule tables, we first build five encoders for 5 fields and convert the corresponding field values into interval IDs. The destination address interval IDs can correspond to the matched routing behaviors and possibility bitmaps. Other fields interval IDs can correspond to the associated possibility bitmap. By using these interval IDs, we can record and query the rule behaviors efficiently in hash tables. Also, the possibility bitmap can reduce the unneeded hash table accesses. Finally, we can obtain the network-wide behaviors by combining the routing behaviors and rule behaviors.

As we encode the field values of packet headers separately, we can avoid the memory explosion that decision-tree based schemes may happen. Also, we use hashing method to record the rules instead of cross-products so that the memory consumption of REHT can be small while the classification speed can be fast.

References

[1] B. Vamanan, G.Voskuilen, and T. Vijaykumar, “EffiCuts: Optimizing Packet Classification for Memory and Throughput,” in *ACM SIGCOMM*, pp. 207-218, 2010.

[2] D. E. Taylor, “Survey and Taxonomy of Packet Classification Techniques,” in *ACM Computing Surveys*, vol. 37, no. 3, pp. 238-275, Sep. 2005.

[3] Y.-K. Chang and Y.-C. Lin, "Dynamic Segment Trees for Ranges and Prefixes," IEEE Transactions on Computers, VOL. 56, NO. 6, pp. 769-784, June 2007.

[4] P. Gupta and N. McKeown, “Packet Classification Using Hierarchical Intelligent Cuttings,” in *Proceedings of IEEE High-Performance Interconnects*, pp. 34-41, 1999.

[5] T. V. Lakshman and D. Stiliadis, “High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching,” in *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 203-214, 1998.

[6] P. Gupta, and N. McKeown, “Packet Classification on Multiple Fields,” in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 147-160, 1999.

[7] S. Singh, F. Baboescu, G. Varghese, and J. Wang, “Packet Classification Using Multidimensional Cutting,” in *Proceedings of ACM Special Interest Group on Data Communication*, pp. 213-224, 2003.

[8] R.E. Bryant, “Graph-based algorithms for boolean function manipulation,” *IEEE Transactions on Computers*, pp.677–691, 1986.

[9] A. Srinivasan, T. Ham, S. Malik, and R.K. Brayton, “Algorithms for discrete function manipulation,” in *IEEE ICCAD*, pages 92–95, 1990.

[10] H. Z. Wang, C. Qian, Ye Yu, H. K. Yang and Simon S. Lam, “Practical network-wide packet behavior identification by AP classifier,” in *IEEE/ACM Transactions on Networking*, vol. 25, pp. 2886-2899, 2017.

[11] T. Inoue, T. Mano, K. Mizutani, S. Minato, and O. Akashi, “Fast packet classification algorithm for network-wide forwarding behaviors,” in *2018 Computer Communications*, vol. 116, pp. 101-117.

[12] Y. K. Chang, C. C. Su, Y. C. Lin, and S. Y. Hsieh, “Efficient Gray Code Based Range Encoding Schemes for Packet Classification in TCAM”, *IEEE/ACM Transactions on Networking*, pp. 1201-1214, 2013.

[13] Y. K. Chang, Y. S. Lin, and C. C. Su, “A High-Speed and Memory Efficient Pipeline Architecture for Packet Classification,” *Proc. the International IEEE Symposium on Field-Programmable Custom Computing Machines* (*FCCM*), pp.215 - 218, 2010.