HSM: A Fast Packet Classification Algorithm

Bo Xu, Research Institute of Information Technology (RIIT), Tsinghua University
Dongyi Jiang, Juniper Networks, Inc.
Jun Li, Research Institute of Information Technology (RIIT), Tsinghua University

***Abstract* – Packet classification on multiple header fields is one of the basic techniques used in network devices such as routers and firewalls, and usually the most computation intensive task among others. To determine what action needs to be taken to a packet, a network device responsible for packet classification must identify the packet's property, such as associated packet flow, based on multiple fields of its header. Fast packet classification on multiple fields is known to be difficult mathematically and expensive practically. In this paper, we describe and discuss a fast packet classification algorithm using a multiple stage reduction scheme similar to the previously well-know algorithm RFC. This hierarchical space mapping (HSM) algorithm requires much less memory usage than RFC while keeps average search time on the same order. HSM has been proved to be very effective with commercial products in real networks.**

***Index terms*—packet classification, access control, security policy, packet filtering, range segmentation, space mapping.**

I. INTRODUCTION

There are many network services that require header based packet classification, especially in policy enforcement by packet filtering. Layer 3 routers make their routing decision as where to forward an arriving packet by checking its destination address in its header against a set of forwarding rules. The rules classify network traffic into flows and predefine the next hops for packets matching the rules. Firewalls take actions to a received packet by searching almost all fields of its header against security policies (also called ACL or access control list in edge routers). Services such as bandwidth management, traffic provisioning, and utilization profiling also depend on packet classification. Action taken to a packet is usually based on the result of classification that is in turn based on a set of filters. In this paper, we use the term "policy" and "action" to describe filters or rules and their associated directives. We also call the collection of policies and policy search as "policy table" and "policy table lookup" rather than the terminology of classifier, and rule search or filter matching.

Multiple field packet classification has two distinct characteristics that make it a classification problem hard to be solved efficiently: the policy table cannot be sorted or cached since the policies in the table are ordered and may have overlapped field values. When a packet arrives to the interface of a network device, there could be multiple policies that match the specified packet header fields, and only the action associated to the policy with the top priority is taken. Policy priority can be calculated based on a defined cost function. In this paper, we assume the priority of each policy is the order it resides in the policy table, a multiple dimension lookup table. Therefore, the first policy in the policy table has the highest priority.

Network environment is very dynamic. Network configuration varies as employee assignments, organization structure, and business relationship changes constantly. Interaction between network services or functional blocks, and intelligence built-in network devices also bring dynamic updates into packet classification consideration. For example, during product and service price information update of a sales database, system administrator may want to set a temporary policy to block all access from business partners to the related database servers. The policy to deny access to the servers will be added on the top of the policy table, as usually table lookup is done top down and for security reason there is always a default policy at the bottom to deny all traffic. Obviously, the table cannot be sorted. For example, it we sort the policy table by destination addresses, individual policies that allow each server to be accessible may get on top of the temporary policy and the blocking will then fail in this case.

The policies can have overlaps in one or more fields, and thus the relationship among the classification tasks can be a graph with cycle, not necessarily a tree. Therefore, the execution order of the classification tasks strictly determines the actions to be taken to the packet. This is why the tasks, and in turn the policies, cannot be cached. For example, all employees working in engineering group E are allowed to access a source code server C, excluding hardware contractor A. However, the access should have lower priority than VoIP traffic and overall traffic load should not exceed a certain bandwidth limit, except release engineer R who has highest priority and best available bandwidth to C. In terms of security policy, the requirement described above can be presented as the following:

a) A's access to C should be denied;
b) R's access to C should be allowed with highest priority and maximum available bandwidth;
c) E's access to C should be allowed with bandwidth limitation.

If c) is cached, a packet stream for A to access C can be passed as it satisfies c), although it will be denied by a) if c) is not cached. The same applies to b) in this example.

Dongyi Jiang and Jun Li worked for ServGate Technologies, Inc. when the initial research and implementation was conducted.

For IPv4, its packet header syntax that is related to classification can be specified as up to 8 fields: 32-bit source and destination network layer (layer 3 or L3) IP addresses, 16-bit source and destination transport layer (layer 4 or L4) TCP or UDP port numbers, 8-bit type-of-service (TOS) field, 8-bit L3 IP protocol field, and 8-bit L4 TCP or UDP protocol flags. If we consider tunneling protocols, there could be more fields such as VLAN tag, etc. In IPv6, its header has similar structure with larger IP address range (128-bit source and destination IP addresses).

Mathematically, the multiple field classification is a point location problem in multi-dimensional space. In computational geometry, this means finding the object that contains of a query point, given a set of geometric objects. It has been claimed that if the objects are none-overlapping (also called non-intersecting or disjoint) fat objects, the best known the computational complexity bounds for n objects (policies in our case) and k dimensions (fields in our case) measures, for $k > 2$, are $O(\log n)$ in time with $O(n^k)$ in space, or $O(\log^{k-1} n)$ in time with $O(n)$ in space [1]. This is impractical as many papers pointed out. For the case of 1,000 firewall packet filtering policies that inspect 4 header fields (source and destination addresses and ports), thus n=1,000 and $k = 4$, n^k means 12TB for IPv4, while $\log^{k-1} n$ means almost 1,000 times memory accesses.

Multiple field classification problems have attracted great attention in recent years due to increased demand of high-end packet forwarding and filtering network devices. The most significant applications are Gbps and faster firewalls. It requires packet filtering capability better than 4 million packets per seconds each direction in full duplex mode. Consider commercial network services should guarantee minimum bandwidth provided to customers rather than promising an average bandwidth that could be choky at times, all discussion in this paper considers the worst case as well as average case. The goal of the study is to find an algorithm that classifies packet at high packet rate with modest storage requirement.

II. PREVIOUS WORK

The design of packet classification algorithms is encumbered by worst-case bounds on search time and memory requirements that are so onerous as to make brutal force algorithms unusable [3]. Therefore, it will be infructuous to attempt to find an algorithm performing well under all circumstances. Research work is mainly oriented to exhuming inherent structures or characteristics of certain classification problems that can make heuristic algorithms that compute “fast enough” and occupy “not too much” memory.

Historically, most packet filtering firewalls use linear search algorithms when performing policy lookup. These algorithms are very time consuming and without upper bound of searching time—the searching time increase linearly as the policy table size grow.

Many research results have been published in recent years to improve the efficiency of firewall policy lookup, essentially solving the problem of multiple field packet classification.

V. Srinivasan, et al, proposed *Grid-of-tries* and *Cross-producting* [3]. Grid-of-tire uses a trie-based data structure, like Hierarchical tries and Set-pruning tries [4], but the adoption of switch pointers to avoid the time-consuming back tracking search makes Grid-of-tries superior to other trie-based algorithms. Cross-producting divides the search space according to the rule segmentations along each dimension. Each segment refers to a sub-region in one of the F dimensions, and the cross-product of the F sub-regions makes up of a sub-space. Search can be done quickly by parallel lookups on each dimension and indexing into the cross-product table but this algorithm bear large space complexity (memory requirement). Several papers followed this direction and proposed improvement or extensions to the Grid-of-tries algorithms [7, 10].

P. Gupta and N. McKeown proposed two algorithms *RFC* [5] and *HiCuts* [6]. RFC can be seen as a form of Cross-producting but is improved by significantly compressing the cross-product table. RFC simplifies packet classification scheme by reducing structure redundancy in the classification process and does not obtain the classification result through one table lookup. The main idea of RFC is to place the smaller cross-products into equivalence classes before combing them to form larger cross-products. This equivalence of partial cross-products considerably reduces memory requirements, because several original cross-product items map into the same equivalence class. HiCuts is an optimized algorithm based on a decision tree structure. At each tree node, the current search space is equally divided along a chosen dimension. The dimension to be cut and the number of cuttings depend on the characteristics of the rules belong to the node. Experimental results show that HiCuts performs well with non-overlap rules but consumes much time and space with overlapped classifiers. HiCuts is not stable at time and space while RFC performs stably at lookup time. HyperCuts [11] was proposed to divide along two dimensions at the same time.

In another direction of packet classification algorithm research, not only effort has been made to exploit structural characteristics, but also introduce of additional heuristic information based on statistical characteristics such as traffic flow [8, 9]. They improved the average case performance and time/space tradeoff of the algorithms described earlier, but the worst case stays unchanged.

Generally, performance of different algorithms can be evaluated with two aspects: theoretical analysis of worst-case complexity and experimental comparison of mostly average cases. **Table.1** is a list of worst-case analysis of aforementioned algorithms, from which we cannot easily conclude which algorithm is superior to the others. But when concentrating on time complexity, we will find that RFC and HiCuts are superior to the other algorithms, while

theoretical analysis and experimental results tell us RFC is more stable in lookup time than HiCuts (RFC needs only 9 memory accesses during one lookup process no matter how the rules distribute).

This paper introduced a new algorithm called HSM. Similar to RFC, HSM used hierarchical space mappings. However, HSM consumes much less memory space while its average lookup time is on the same order as RFC. Experimental results will be given afterwards in the paper and HSM has been proved to be very effective in real networks by commercial products of ServGate Technologies, Inc.

Algorithm	Worst time	Worst space
Linear Search	$O(N)$	$O(N)$
Grid-of-tries	$O(W^{F-1})$	$O(N)$
Cross-producting	$O(FW)$	$O(N^F)$
RFC	$O(F)$	$O(N^F)$
HiCuts	$O(F)$	$O(N^F)$

Table.1 Worst-case complexity comparison of algorithms. In this table N is the number of rules, W is bit-width of a certain dimension (e.g. for IPv4 IP address, W=32), F is dimensionality of the search space.

III. HIERARCHICAL SPACE MAPPING ALGORITHM

Many previous works assume that some or even all fields in a policy are presented as prefix. In contrast, most inspected fields or matching templates in a policy are represented as numbers or ranges naturally, except networks (subnets). Even network addresses such as addresses of a group of servers are easier to be represented as ranges sometimes. As range-to-prefix conversion can generate a large number of policies and thus increase computational workload of policy lookup, ranges are used in this paper rather than prefixes. When needed, the one time prefix-to-range conversion does not add more policies and therefore has no impact on packet classification.

The algorithm proposed in this paper can be applied in general cases of multiple field classification problems where sorting and caching do not help. However, the discussion in this paper focuses on IP network application, using a four-tuple firewall security policy lookup as example. The four-tuples considered in the firewall policy lookup example are destination address (DA), source address (SA), destination port (DP), and source port (SP). Normally, packet filtering inspects at least five-tuples. Here it is assumed properties other than the four tuples will be handled before or after policy lookup, such as TCP and UDP will have two separate policy tables or checked separately after the four-tuple search.

The basic idea of the HSM algorithm proposed in this paper is to reduce the searching fields by mapping the lookup domains two-to-one, step by step and hierarchically. First, it maps the 2 IP address spaces (DA, SA) and the 2 port number spaces (DP, SP) into non-overlapped segments precisely according the network address ranges and port number ranges used in the policy table, and reduces the original four-dimension space to a two-dimension space by looking up the following two tables:

AMT — source/destination IP address mapping table
PMT — source/destination port number mapping table

Second, the two-dimension space resulted from the previous step is transformed to the one-dimension policy space. This is done by looking up the third table:

PLT — policy lookup table

Figure.1 gives us an overview of the packet flow in HSM algorithm. We will elaborate the whole process step by step afterwards.

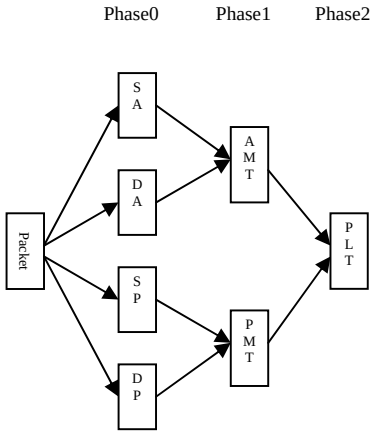
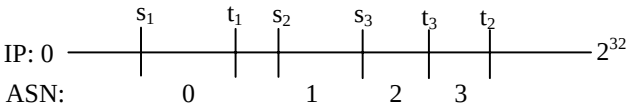


Figure.1 Packet flow in HSM

IP Address Fragmentation

IP address fragmentation is done for both SA space and DA space respectively but in the same way. For each address range (including address or subnet) appeared in the policy table, its two boundary IP addresses are marked down in the corresponding SA or DA IP space. When this is finished for each and every policies in the policy table, for each segment that has at least one policy falls in it, an address sequence number (ASN) is assigned in the ascend order along the increasing IP address, starting from 0. **Figure.2** is an example that illustrated the IP address segmentation.



s: starting address of an address range; t: ending address of an address range.

Figure.2 IP address segmentation

There are many ways to map a given IP address (i.e. the source or destination IP address of a received packet) to a segment. In HSM, this is achieved by maintaining a balanced binary tree.

Port Number Fragmentation

The principle of port number fragmentation to get port sequence number (PSN) is quite similar to that of IP address fragmentation.
For the port number mapping, a direct table (2^{16} or 65536 in size) lookup is feasible when there is enough memory that can be allocated for it and usually more efficient.

Lookup Table Structure

The AMT table is a two-dimension table with the source address sequence number (SASN) and the destination address sequence number (DASN) as the indexes, and filled with address group numbers (AGN). The PMT table is similar to AMT except using the source port sequence number (SPSN) and the destination port sequence number (DPSN) as the indexes of the two dimensions, and filled with port group numbers (PGN). The PLT is a two-dimension table with the AGN and PGN as the indexes, and filled with policy order numbers representing priority of the policies.

Lookup Table Setup

AMT
When putting IP address segmentation, we assign a bitmap (BM) for each ASN indicating which policies in the policy table contain this ASN. This bitmap has one bit for each policy in the policy table. For example, if the policy table contains 8 policies and SA#1 is covered by policy#1, policy#2 and policy#6, then a BM of 01000110 will be set to SA#1.

Each entry of AMT is given an address group number (AGN) according to the order of its appearance, along with a BM tagged to it. The BM is formed by an OR operation of the two BMs of SA and DA. We can use **Table.2** as an example. Assume that SA#1 has bitmap 01000110 and DA#0 has bitmap 00100110, then the combination of the two comes to bitmap 00000110. So we put 1,2 in the entry corresponding to SA#1 and DA#0. Different AGN has different BM. If the result of combination is the same as previous AGN, then the AGN and associated BM for the entry will stay unchanged.

PMT
The process of PMT is almost like AMT. We also combine the BM of each PSN indicating the policies contain it. Each entry of PMT is given a port group number (PGN) according to the order of its appearance, and has a BM tagged to it. The BM is formed by an OR operation of the two BMs form SP and DP. **Table.3** provides a reference and the foundation process is just like **Table.2**.

Note that the BMs are not physically stored in lookup table; they are only used in the setup of lookup tables and will be released after PLT established.

PLT
Table.4 shows an example of PLT, which is generated from **Table.2** and **Table.3**. The table is 5 by 7 in size because **Table.2** engages 5 different AGN with BM indicating policy sets: {1,2}, {1}, {0}, {0,1,2}, {2} and **Table.3** engages 7 different PGN with BM indicating policy sets: {1}, {0}, {0,1},{0,1,2}, {0,2}, {1,2}, {2}.

Each entry of PLT is filled with a policy number. We combine the BMs of AGN and PGN, and then pick out the policy number of the highest priority. For example, AGN#0 has a BM indicating policy sets {1,2} while PGN#2 has a BM indicating policy sets {0,1,2}, then the result of the combination will be rule sets {1,2}. We fill the entry corresponding to AGN#0 and PGN#2 with policy 1 because of its higher priority.

AMT	SA#0	SA#1	SA#2	SA#3
DA#0		1,2	1,2	1
DA#1	0	0,1,2	1,2	1
DA#2		2	2	

Policy 0 falls into SA segments 0 through 1, and DA segment 1; policy 1 falls into SA segments 1 through 3, and DA segments 0 through 1; policy 2 falls into SA segment 1 through 2, and DA segments 0 through 2.

Table.2 AMT structure and setup

PMT	SP#0	SP#1	SP#2	SP#3
DP#0		1	1	
DP#1	0	0,1	0,1,2	0,2
DP#2		1	1,2	2

Policy 0 falls into all SP segments, and DP segment 1; policy 1 falls into SP segment 1, and all DA segments; policy 2 falls into SP segments 1 through 2, and DP segments 1 through 2.

Table.3 PMT structure and setup

PLT	AGN#0	AGN#1	AGN#2	AGN#3	AGN#4
PGN#0	1	1		1	
PGN#1			0	0	
PGN#2	1	1	0	0(1)	
PGN#3	1(2)	1	0	0(1,2)	2
PGN#4	2		0	0(2)	2
PGN#5	1(2)	1		1(2)	2
PGN#6	2			2	2

Table.4 PLT structure and setup

Policy Lookup for Packet Classification

1. Parse the header of a received packet to get DA, SA, DP, and SP;
2. Travel corresponding balanced binary trees to get the DASN and SASN according to DA and SA, respectively;
3. Travel corresponding balanced binary trees or lookup corresponding tables to get the DPSN and SPSN according to DP and SP, respectively;
4. Lookup AMT to get AGN by using the (DASN, SASN) pair as indexes;
5. Lookup PMT to get PGN by using (DPSN, SPSN) pair as indexes;
6. Lookup PLT to ultimately find the policy number by using (AGN, PGN) pair as indexes.

A Simple Example of HSM
We present a simple example of a HSM that showing the complete HSM operation including preprocessing to set up the tables based on a given policy table and to perform policy lookup to determine the actions to the packet under inspection. The example is shown in **Figure.6** in the Appendix, which is based on a four-tuples searching case of **Table.7**, also in the Appendix.

IV. Experimental Results

Theoretical Analysis
Assume we have N policies, the height of balanced binary tree used to hold all the boundary information of IP segments is $\log(2N+1)$. A policy lookup in F fields will only need to travel through at most F balanced binary trees for compare and branch, then lookup $\log(F-1)$ tables. Therefore, the worst case computational complexity of policy lookup is $O(\log N)$ in time. In general, a F-tuple HSM search with N policies will have time complexity of $O(F\log(2N+1)+\log(F-1))$, or $O(\log N)$ when $2<F<<N$.

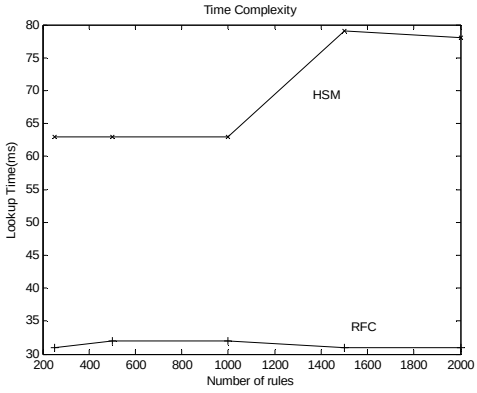
Using the four-tuple search described earlier as an example. Assuming we have a case of 1024 policies. The SASN and DASN trees are of maximum height 11. To travel these two trees, we need maximum 22 times compare and branch. If we use the direct table lookup to determine the port segment, only 2 times memory access is needed. To index through AMT, PMT and PLT, three times of table lookup are needed, which means 3 times of memory access. Totally, we need maximum 22 compare and branch plus 5 table lookups to locate the right policy by HSM.

When there are N policies, the maximum size of PLT is $O(N^4)$, and the worst case size of AMT and PMT are both $(2N+1)*(2N+1)$, as there could be at most $2N+1$ IP address or port number segments for N entries in a policy table.

However, with real-life policy tables, there is structural redundancy among policies as observed by Gupta [5, 6]. In the example shown in the Appendix where N=3, the size of AMT and PMT are both 4 by 3, rather than $(2N+1)*(2N+1)$ or 7 by 7, and the size of PLT is 5 by 7, rather than in the order of $N^2 * N^2$ or 49 by 49. Therefore, due to the elimination of overlapping, memory requirement in practice (average cases) are far less than theoretical worst cases.

Experimental Results
Experimental results are shown in **Figure.3** and **Figure.4**, which provide comparisons of RFC and HSM on average lookup time and memory occupied. The testing policy table is a real-life ACL from Tsinghua University, Beijing, China.

From **Figure.3**, we can see that lookup time of HSM is slower than RFC, and the time has little change when the number of rules grows. This result is consistent with theoretical analysis. We know that RFC needs 9 memory accesses during one lookup process [5] regardless of number of policies, while HSM needs $4*\log(N) + 3$ memory accesses for N policies. When N is on the order of 1,000, HSM average lookup time is determined by 15~19 memory accesses.



Total lookup time of 122,936 packets.
Figure.3 Comparison of average lookup time

From **Figure.4** we can see that HSM is superior to RFC in space complexity. With the number of policies grows to above 2 kB, HSM can save more than 3 MB fast memory space compared to RFC.

	Number of rules	Memory use of RFC(k)	Memory use of HSM(k)
FW1	69	797	41
FW2	341	910	262
CR1	1001	1,666	923
CR2	2180	13,840	10,062

Table.5 Comparisons of memory occupied

For more test to evaluate performance of HSM algorithm, we managed to obtain 4 real-life policy tables from enterprise networks and major ISPs. The two firewall policy tables are named FW1, FW2, the two router access control lists are called CR1, CR2. **Table.5** shows us a comparison between HSM and RFC on memory occupied.

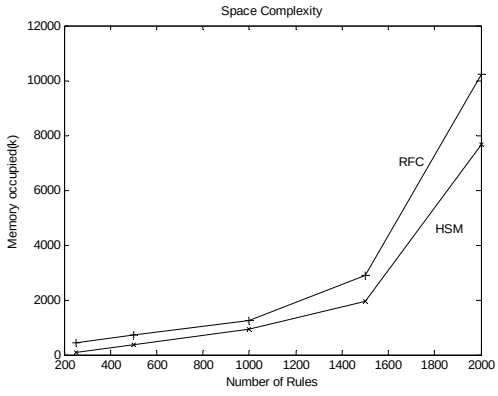


Figure.4 Comparison of memory requirement

We can explain the efficiency in space requirement on the structure level. **Figure.1** has shown us the packet flow in HSM, while the following **Figure.5** will give us the logical chart of RFC.

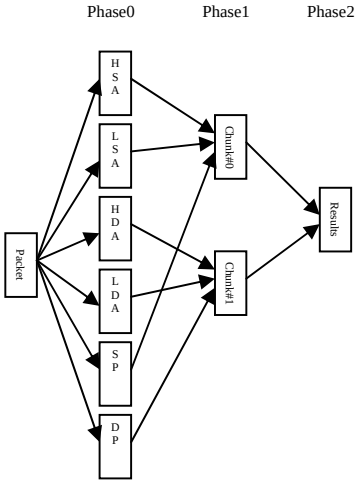


Figure.5 Packet flow in RFC

In **Figure.5**, HSA indicates the high 16 bits of Source Address and LSA stands for the low 16 bits of Source Address. The same formula can be extended to HDA and LDA. We consider the same rule sets in comparison between RFC and HSM. Assume that in RFC, HSA has *R1* segments, and that LSA has *R2* segments, and HDA *R3* segments, LDA *R4* segments, SP *R5* segments, DP *R6* segments. Meantime, we suppose that in HSM, SA has *H1* segments, DA *H2* segments, SP *H3* segments, DP *H4* segments.

Obviously, *R5* is equal to *H3* and *R6* is equal to *H4*. And it can be concluded that $R1+R2 >= H1$ and $R3+R4 >= H2$. We have $R1*R2 > R1+R2$ and $R3*R4 > R3+R4$ when $R1, R2, R3, R4 > 2$. So we get the inequations $R1*R2 > H1$ and $R3*R4 > H2$, from which it results to $R1*R2*R5 > H1*H3$ and $R3*R4*R6 > H2*H4$. Thus we get the point that HSM requires less memory than RFC in *phase1*. And we have described before that HSM used trees in *phase0* rather than chunks, it only stores the start and end IP addresses of fragments. So it needs much less memories than RFC. The two decreases in memory determine that HSM is superior to RFC in space consuming, for the memories occupied in *phase2* are just the same, that is because the segments of the two chunks of *phase1* must be the same.

Comparison with Other Packet Classification Schemes

Table.6 shows a qualitative comparison of some of the schemes for packet classification.

Schemes	Advantages	Disadvantages
Grid-of-trie	Good storage requirements and fast lookup rates for two fields. Suitable for large policy tables.	Not easily extendible to more than two fields. Not suit-able for non-contiguous masks.
Corss-producting	Fast accesses. Suitable for multiple fields. Can be adapted to non-contiguous masks.	Large memory requirements. Suitable without caching for small policy tables.
RFC	Suitable for multiple fields. Works for non-contiguous masks. Fast lookup rate.	Large preprocessing time and memory requirements for large policy tables.
HiCuts	Suitable for multiple fields. Performs well for non-overlapped rules. Good tradeoff between time and space.	Large preprocessing time and memory requirements for large policy tables. Not stable for different policy lookups.
HSM	Suitable for multiple fields. Fast lookup rate. Reasonable memory requirements for real-life policy lookup.	Large preprocessing time and not small memory requirements for large policy tables.

Table.6 Comparison of popular algorithms

Perspective on IPv6

The development of NGI and the proposition of IPv6 set new challenges to packet classification algorithms. The present tree-based algorithms like Grid-of-trie and HiCuts will encounter big problems. The increasing of tree depth would result in the boom in storage requirements and make the policy lookup time-consuming. Cross-producting and RFC might not be effective any more because of oppressive space needed. But the proposed algorithm in the paper will perform well under IPv6 structure, for it is the fragments of SA, DA, SP, and DP that determines the storage requirements. The space needed increases when the numbers of fragments increase, but would not change a lot when the IP address bit length increases. It just needs more memory to store the longer IP addresses in *phase0*.

The HSM algorithm was successfully implemented in wire-speed giga-bit security gateway products commercialized by ServGate Technologies, Inc. The system was built based on two Intel IXP1200 network processors each handling inbound and outbound network traffic, respectively. Test was carried out with line-rate packet flows generated by Spirent SmartBit to pass through the system, with the percentage of denied traffic (packets dropped by the action of the last policy which denies all packets not explicitly permitted by any other policies) increasing from 0% to 100% cross all different packet sizes and various policy table sizes. It was shown that the packets allowed by the explicitly defined policies, which permit the respective packet flows, passed the system almost unaffected, meaning the policy lookup is not the bottleneck. Note that session lookup or stateful inspection was implemented in the system so that allowed traffic flows had almost no load on policy lookup, except the first packet of each flows. In cases where policy lookup is the bottleneck, as denied traffic becomes heavy, processing power of the systems would be drew too much to perform policy lookup for denied traffic as each packet rather than flow need to go through packet lookup individually, and therefore packets are dropped randomly, causing throughput degraded faster than linear.

V. CONCLUSION

Due to the characteristics of ordered and overlapping policies, packet classification on multiple fields cannot be expedited by policy sorting prior to policy lookup or policy cache during policy lookup. To achieve high-performance policy lookup, special hardware and associated algorithms can be applied but they introduce additional cost. Those hardware solutions are usually neither flexible nor scalable. The HSM algorithm proposed in this paper provided a novel generic solution that can be implemented either in software or hardware, with reasonably balanced time and space computational complexity.

Future work can be conducted in optimization of date structure to further improve lookup and update efficiency by leveraging on modern network processor architecture.

VI. ACKNOWLEDGEMENT

The authors would like to acknowledge ServGate and related engineers of the company for the support and contribution to the study and experiment. Thanks also due to Dr. Enke Chen and Mr. Yaxuan Qi, for their help in providing data and contributing their thoughts.

VII. REFERENCE

[1] M.H. Overmars and A.F. van der Stappen, Range Searching and Point Location Among Fat Objects, Journal of Algorithms, Vol. 21, No. 3, 1996.

[2] David E. Taylor, Survey & Taxonomy of Packet Classification Techniques, Technical Report of Washington University in Saint-Louis, 2004.

[3] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, Fast and Scalable Layer Four Switching, Proc. ACM SIGCOMM, 1998.

[4] P. Gupta and N. McKeown, Algorithms for Packet Classification, IEEE Network, 2001.

[5] P. Gupta and N. McKeown, Packet Classification on Multiple Fields, Proc. ACM SIGCOMM, 1999.

[6] P. Gupta and N. McKeown, Packet Classification Using Hierarchical Intelligent Cuttings, Proc. Hot Interconnects, 1999.

[7] F. Baboescu, S. Singh, and G. Varghese, Packet Classification for Core Routers: Is There an Alternative to CAMs? Proc. INFOCOM, 2003.

[8] T.Y.C Woo, A Modular Approach to Packet Classification: Algorithms and Results, Proc. IEEE INFOCOM, 2000.

[9] Y.X Qi and J. Li, Dynamic Cuttings: Packet Classification with Network Traffic Statistics, submitted to Proc. INFOCOM, 2004

[10] L. Qiu, G. Varghese, and S. Suri, Fast Firewall Implementation for Software and Hardware Based Routers, Proc. ICNP, 2001.

[11] F. Baboescu and G. Varghese, Packet Classification Using Multidimensional Cutting, Proc. ACM SIGCOMM, 2003.

VIII. APPENDIX

Rule	SA Range	DA Range	SP Range	DP Range	Action
(0)	0.0.0.0~64.0.0.0	32.0.0.0~64.0.0.0	0~65535	128~256	deny
(1)	32.0.0.0~255.255.255.25	0.0.0.0~64.0.0.0	64~256	0~65535	permit
(2)	32.0.0.0~128.0.0.0	0.0.0.0~255.255.255.255	128~65535	128~65535	deny

Table.7 Policy table used for example

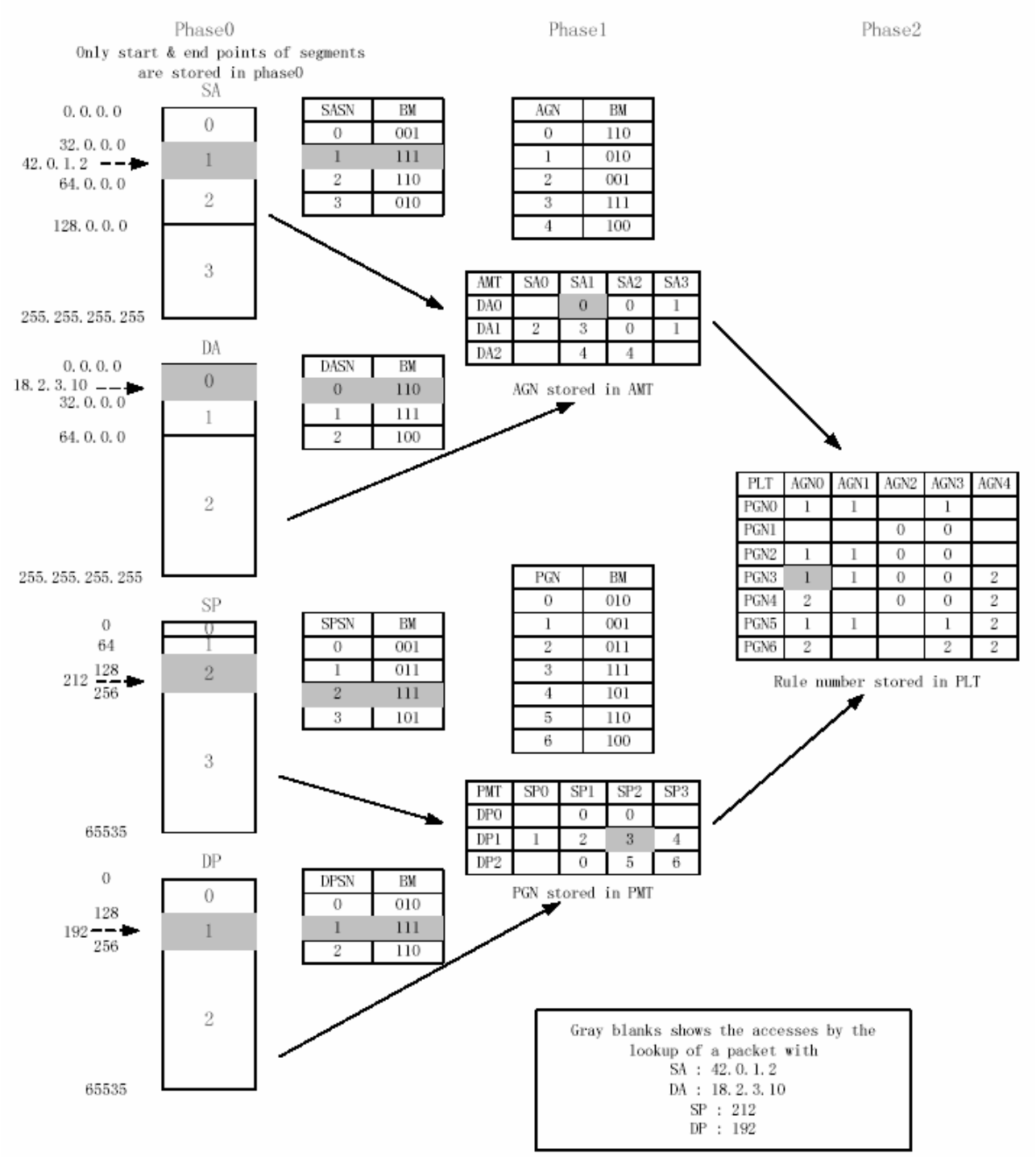


Figure. 6 This figure shows the contents of HSM tables for the example of **Table.7**.
The sequences of accesses made by the example packet have been shown in grayed blanks.