Hashing Functions Performance in Packet Classification

Mahmood Ahmadi and Stephan Wong
Computer Engineering Laboratory
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
{mahmadi, stephan}@ce.et.tudelft.nl

Abstract— **Packet classification remains an important aspect of network processing as it encompasses increasingly more functionality due to newly introduced services. Essentially, it entails the matching of incoming packets against a database of rules performing the operation that is associated with the matching rule with the highest priority. Within the hashing-based packet classification algorithms, the tuple space search algorithm is gaining much interest. Using tuple spaces, the rules can now be subdivided into sets and in turn these sets can be searched in parallel. Within each set, the hashing functions determine the location of storing the rules. However, depending on the chosen collection of hashing functions, rules (within a set) can be mapped to the same location (containing multiple buckets to store the rules) resulting in a *collision*. A side-effect of such collisions is that more memory accesses are needed to resolve the collision resulting in degraded performance. In this paper, we compare and evaluate different hashing functions taking from the H3 class of hashing functions using which collisions can be reduced and in turn reduce the average bucket sizes. Our results show that when using the H3 class of hashing functions, we were able to reduce the number of collisions by at least 7% and by at most 49% when compared to other hashing functions.**

Keywords: Packet classification, tuple space, universal hashing function, collision

I. Introduction

Traditionally, packet classification entailed the forwarding of packets solely based on the destination address that is specified in one of the many header fields within a packet. However, the importance of packet classification has increased greatly with the introduction of services like Quality of Service (QoS), virtual private network (VPN), policy-based routing, traffic shaping, firewalls, and network security. Additional criteria had to be satisfied by taking into account additional header fields from the packet, such as, source address, protocol type, source and destination port numbers (see Figure 1). Packet classification can be seen as the categorization of incoming packets based on their headers according to specific criteria that examine specific fields within a packet header. The criteria are comprised of a set of rules that specify the content of specific packet header fields to result in a match [3][4][9]. Traditionally, hashing is utilized in packet classification to speed up the process of determining whether an incoming packet matches a certain rule (that in turn determine the action to take on the packet). Furthermore, certain rules only examine specific fields (sometimes even only the prefix) of the packet headers specified using a tuple. Consequently, rules that examine the same fields (or the prefix) are combined in a so-called tuple space with all those rules hashed into a hash table. Summarizing, the (prefix of) headers of incoming packets are selected for each tuple (the number of tuples depends on the rule set) and hashed to determine whether it matches a rule within that particular tuple space stored in the hash table. A common problem in using hashing is collision, i.e., the mapping of rules in the hash table can be to the same hash table location. Consequently, when an incoming packet is hashed to a hash table entry containing multiple rules it must be matched to all these rules resulting in much longer processing time. Therefore, the choice of hashing functions will have a certain impact on the number of collisions.

In literature, hashing functions such as belonging to the H3 class have been evaluated for random data by random selection of hashing functions [6]. As mentioned before, rules can be categorized in the tuple-space after which hashing functions are used to determine the storage location of these rules. In this paper, we propose to utilize the H3 class of hashing functions to decrease the number of mentioned collisions and to design an adaptive software packet classifier. In particular, we derive a hashing function through the H3 class of hashing functions in order to demonstrate its effect on the number of collisions and the average bucket size. More specifically, we investigate several well-known hashing functions to determine the number of collisions and show that the number of collisions can be further reduced by at least '7%' and by at most '49%' for real rule-set databases by utilizing class $H3$ hashing functions. It also decreases the average bucket size (a buffer that stores multiple rules) in compared to other hashing functions. The paper is organized as follows: Section II describes related work. Sections III represents the tuple space algorithms for packet classification, the definition of the class $H3$ hashing functions, and packet classifier architecture. Section IV presents related results. Section V, we draw the overall conclusions.

II. Related Work

In this section, we present different related works by others in packet classification algorithms that utilize hashing functions. In [6], the $H3$ class of hashing functions as a class of universal hashing functions with implementations in

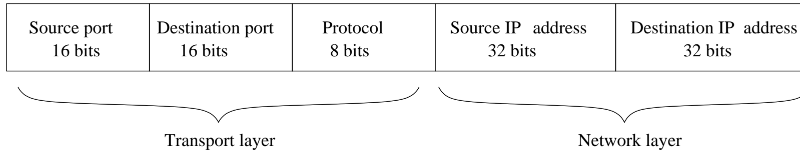

Fig. 1. Most important fields that are used in classification algorithms

hardware, are introduced. The hashing function performance using maximum search length that is called *'llps'* (length of the longest prob sequence) is evaluated. The hashing function selection procedure is random. In [9][11], a packet classification algorithm using tuple space is introduced that concatenates the necessary number of bits from rule-sets and incoming packets to make a hash key. To decrease the number of collisions, the memory size is increased. It also assumes that the hashing function is perfect and the complexity of packet classification algorithm is investigated. In practice, to achieve a perfect hashing function is difficult.

In [8], a Fast Hash Table (FHT) architecture using a Bloom filter for packet classification and IP lookup is presented. FHT increases the memory size to several times using a TCAM memory in order to reduce collisions and memory access times in the hash table and utilizes a class of universal hashing functions where the hashing functions are selected randomly. In our work, for each rule-set database, we find an $H3$ hashing function via searching through one thousand hashing functions from the $H3$ class of hashing functions. This function is utilized by our software packet classifier that stores rules in a hash table. The software packet classifier decreases the number of collisions by achieving a hashing function with a minimum number of collisions and consequently improves the packet classification speed and throughput.

III. Tuple Space Packet Classification and Hashing Functions

In this section, the concept of the tuple space packet classification algorithm, related hashing functions, and a software packet classifier are presented.

A. Tuple Space Classification

A high level approach for multiple field search employs tuple space. A tuple defines the number of specified bits in each field of the rule. Srinivasan, et. al. introduced the tuple space approach and the collection of tuple search algorithms in [9][10][11]. We provide an example of rule-set for rule classification on five fields in Table I.

In Table I, words 'eq' and 'gt' are abbreviations of operators representing equal to and greater than a port number in the expressed rule. Address prefixes cover 32-bit addresses and port ranges cover 16-bit port numbers, for address prefix fields, the number of specified bits is simply the number of non-wildcard bits in the prefix, for the protocol fields the value is simply a boolean: '1' if a

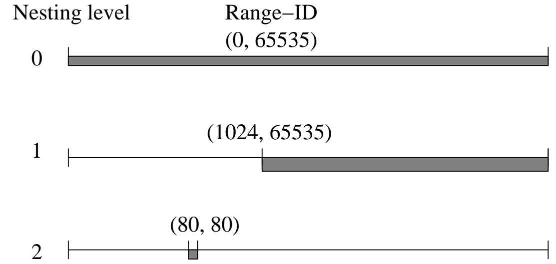

Fig. 2. Assigning values for ranges, based on Nesting Level and Range-ID

protocol is specified, '0' if a wildcard is specified [9][11][12]. The number of specified bits in a port range are not as straightforward to define. The authors introduce the concept of nesting levels and Range-ID to define the tuple value for port ranges. The nesting level specifies the layer of the hierarchy and the Range-IDs uniquely labels the range within its layer. In this way, we can convert all port ranges to a (Nesting level, Range-ID) pair. The nesting level is used as the tuple value for the range, and the Range-ID is used to identify the specific range within the tuple. The full range, in this example (0-65535) always has the id '0'. The two ranges at level 1, namely (0, 1023) and (1024, 65535) in our example receive id 0 and 1 respectively. The example of mapping port range to nesting level and Range-ID for Table I is depicted in Figure 2.

For example, a search key for the tuple [8, 24, 2, 0, 1] is constructed by concatenating the first octet of the packet source address, the first three octets of the packet destination address, the Range-ID of the source port, the range at nesting level 2 covering the packet source port number, the Range-ID of the destination port, range at nesting level 0 covering the packet destination port number, and the protocol field. All algorithms employing the tuple space approach involve a search in the complete tuple space or a subset thereof. The tuple space techniques can exploit parallelism [11]. For example rule set in Table I a search key would have to probe 5 tuple instead of searching all 6 tuples. In practice, the use of real rule sets in tuple space search reduced the number of searches by a factor of four to seven [9][11].

B. Hashing Functions

Several kinds of hashing functions are utilized in packet classification: additive, rotative, bit extraction, XOR-based, mixed, and universal hashing functions [5][6]. In additive hashing functions, the hash value is constructed by traversing

Rules	Destination IP (address mask)	Source IP (address mask)	Port No.	Protocol No.	Tuple space
R1	192.168.190.69 (255.255.255.255)	192.168.80.11 (255.255.255.0)	*	*	[32, 24, 0, 0]
R2	192.168.3.0 (255.255.255.0)	192.168.200.157 (255.255.255.255)	eq www	tcp	[24, 32, 2, 1]
R3	192.168.198.4 (255.255.255.255)	192.168.160.0 (255.255.255.0)	gt 1023	udp	[32, 24, 1, 1]
R4	193.164.0.0 (255.255.0.0)	193.0.0.0 (255.0.0.0)	eq www	udp	[16, 8, 2, 1]
R5	192.168.0.0 (255.255.0.0)	192.0.0.0 (255.0.0.0)	eq www	tcp	[16, 8, 2, 1]
R6	0.0.0.0 (0.0.0.0)	0.0.0.0 (0.0.0.0)	*	*	[0, 0, 0, 0]

TABLE I

EXAMPLE RULES AND RELATED TUPLES.

through the data and continually incrementing an initial value by a calculated value relative to an element within the data. The calculation performed on the element value is usually in the form of a multiplication by a prime number. In rotative hashing functions, every element in the data string is used to construct the hash value, but unlike additive hashing the values are put through a process of bitwise shifting. In the bit extraction method, the hashing function entails selecting j bits out of the i bits of the key. In XOR-based hashing functions, the i bit key is partitioned into j bit segments. The segments are exclusive-ORed to produce the hash address. In the mixed method, the hashing functions utilize any or all of the mentioned techniques. It obvious that the performance of these functions dependent on the key set [5][6]. These hashing functions take longer to execute compared to the earlier mentioned functions that only utilize bitwise logical operations.

A solution to achieve a hashing function that is independent from the key set is by utilizing a class of universal hashing functions that exploits bitwise logical operations in their definition. Let H represent a class of functions with input set A and output set B. H is said to be universal if for all x, y in A, no pair of distinct keys collide under more than $(1/|B|)$th of the functions where $|B|$ denotes size of B [2]. A special class of universal hashing functions is called $H3$ hashing functions [2][6]. The $H3$ class of hashing functions are defined as follows: Let $A = 0, 1, 2, ..., 2^i - 1$ be the key space, $B = 0, 1, 2, ..., 2^j - 1$ be the address space, i denotes the number of bits in in the key, j denotes the number of bits in address, Q denotes the set of $i \times j$ boolean matrices, and I represents the given key set, $I = x_1, x_2, ..., x_n$, $I \subset A$. For a given $q \in Q$ and $x \in A$, let $q(k)$ be the $k'th$ row of the matrix q and x_k be the kth bit of x. The hashing function $h_q(x) : A \rightarrow B$ is defined as follows:

$$h_q(x) = x_1.q(1) \bigoplus x_2.q(2) \bigoplus ... \bigoplus x_i.q(i) \qquad (1)$$

where . denotes the binary AND operation and $\bigoplus$ the exclusive OR operation. The hashing function from this class can be easily implemented in hardware. The hardware stores the $i \times j$ boolean matrix that can be organized in a bank of registers [6] where the boolean matrices can be generated in software and then loaded into the bank of registers. Based on the tuple space representation for the rule-set database and IP packets, the size of the input key is 88 bits long (32 bit source IP address, 32 bit destination IP address, 8 bit Range-ID for source port, 8 bit Range-ID for destination port and 8 bit protocol field). The maximum size of the tuple or address space is assumed to be 2^{16} rules for 16 bit address. Therefore, $Q_{88\times16}$ denotes a set of matrices to define the $H3$ class of hashing functions in the tuple space packet classification algorithm.

C. Packet Classifier Architecture

The tuple space packet classification architecture is depicted in Figure 3.

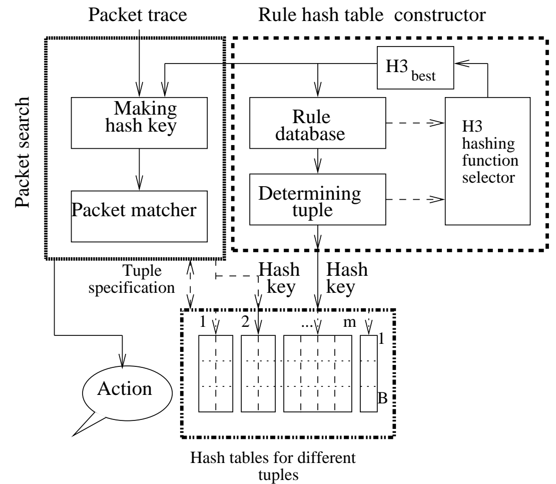

Fig. 3. The architecture of packet classifier with optimized hashing functions in tuple space.

The architecture comprises two main components: rules hash table constructor and the packet search unit. The hash table constructor reads the rules from a rule-set database and extracts the rule specification to determine the corresponding tuple that the rule belongs to. After determining the tuple, the rule should be hashed in the hash table associated with the tuple. Subsequently, the hash key needs to generated utilizing the hashing function after which the rules are stored in the hash table. Before generating the hash key, the "$H3$ hashing function selector"-component determines the best hashing function within the H3 class of hashing functions. This component receives different rules from the

rule database and hash them using different hashing functions from class $H3$ (in here 1000 hashing functions) to derive the hashing function with the minimum number of collisions that is called $H3_{best}$. Afterwards, this hashing function is utilized by the packet search component for the matching of incoming packets against stored rules in the hash tables related to tuples. In the hash table construction, different hash tables with unequal sizes are created since each hash table corresponds to a single tuple that relates to different rules with their own specifications. Usually more than half of the rules belong to two tuples only. This is depicted in Figure 3. In this figure, m denotes the number of tuples and B represents the bucket size.

The packet search component of the packet classifier processes the incoming packets to find the matching rules in the hash tables corresponding to the tuple specifications. Therefore, for each incoming packet, a hash key is extracted based on the tuple specification. This procedure exploits the generated hashing function by the hash table constructor. Consequently, m hash keys are utilized to access the m hash tables (after hashing) to determine whether matching rules can be found. The importance of a hashing function can be seen in this step, since the decreasing one access in hash table construction step will decrease m (number of tuples) accesses in packet search. The accessing of the hash table can be performed in a serial or parallel manner. Finally, the actual packet is checked against the found rules in the packet matcher component. For each packet, the number of hashing operations are equal to the number of tuples in the system or the number of distinct hash tables, therefore, the number of accesses in a sequential search process per packet is equal to the number of tuples.

Due to the insertion of new rules and possible incremental updates, the rule-set database can evolve (change) over time. Therefore, after the nsertion of certain rules, the utilized $H3$ hashing function is no longer desirable since the newly inserted rules may create collision. In this case, we can define an *update threshold*, which is a value that when exceeded by the number of newly inserted rules, causes the packet classifier to execute the hash table constructor component to determine a new $H3$ hashing function. The value of the update threshold is assumed to be the worst case of the improvement rate of the system. In here, based on the result section, the update threshold is assumed to '7%' of number of rules in the rule-set database.

IV. RESULTS

For the system test, we utilized different rule-set databases and packet traces that are used by Applied Research Laboratory in Washington University in St. Louis[7].

In the rule-set database, each rule consists of 5 header fields including "[Source IP address, Destination IP address, Source port, Destination port, Protocol]" and the format is *"@[Source IP address prefix in dot-decimal notation]/[Prefix length] [Destination IP address prefix in dot-decimal notation]/[Prefix length] [Low source port] : [High source port] [Low destination port] : [High destination port] [Protocol value in hexadecimal]/[Protocol mask in hexadecimal]"* [7]. An example of a rule in the rule-set database is : *"@204.152.188.80/28 204.152.188.64/28 67 : 67 67 : 67 0x11/0xff"*.

The packet header trace format is *"[Source IP address in decimal] [Destination IP address in decimal] [Source port value in decimal] [Destination port value in decimal] [Protocol in decimal]"*. an example of a packet header in a packet trace is: *"3337533518 2390673931 65535 65535 1 9"*. The specifications of rule-set databases, and packet traces are shown in Table II. Table II includes seven rule-sets database and packet traces. The rule-sets FW1, ACL1 and IPC1 were extracted from real rule-sets and others were generated by Classbench benchmark [7].

The results are generated using two different load factors. The first load factor has the value of $\frac{n}{prime(n)}$, where n is the number of rules in the tuple and the value of $prime(n)$ represents the size of address space (tuple size). The $prime(n)$ is the first prime number that is larger than n. There is no substantial mathematical work that can definitely prove the relationship between prime numbers and pseudo random number generators. Nevertheless, the best results have been found when using prime numbers. Generating prime numbers for each tuple is time consuming and implementation is not helpful in hardware. The second load factor which is easier to implement in hardware is $\frac{n}{2^{[log_2^n]+1}}$, where the size of address space in each tuple is equal to 2 to the power of number of bits in number of rules in tuples. In these tables, $H3_{best}$ represents a function of class $H3$ hashing functions that result in minimum number of collisions. $H3_{worst}$ represents a function of class $H3$ hashing functions that result in maximum number of collisions. We utilize six general hashing functions that belong to mixed hashing function type as follows: APH, DJBH, DEK, ELF, JS and CRC [1][5]. The specification of mentioned hashing functions are presented in Table III.

Table III includes the name and the number of generated collisions for different rule-set databases with a load factor of $\frac{n}{prime(n)}$. The function APH is the best hashing function among the mentioned mixed hashing functions. Therefore, we only present the results of APH hashing function. The number of generated collisions for different hashing functions using the packet classifier is presented in Table IV.

In Table IV, we can observe that, using the $H3_{best}$ of algorithms decreases the number of collisions by at most '49%' and by at least '7%' in comparison to APH hash function for real rule-set databases. It also shows, that constructing tuple spaces with a load factor of $\frac{n}{2^{[log_2^n]+1}}$, decreases the number of collisions in comparison to a load factor $\frac{n}{prime(n)}$. The average bucket size for different hashing functions with different rule-set databases is presented in Table V.

The average bucket size for nonempty buckets is also evaluated. Some buckets do not include any rules, therefore,

Rule-set database	FW1-100	FW1-1k	FW1-5k	FW1-10k	FW1	ACL1	IPC1
Number of rules	92	971	4653	9311	266	752	1550
Number of tuples	26	42	52	57	36	44	179
Number of Packets	920	8050	46700	93250	2830	8140	17020

TABLE II

RULE SET DATABASE AND PACKET TRACE SPECIFICATION.

they are excluded from the computations. Based on Table V, we can observe that $H3_{best}$ has slower bucket size in comparison to $H3_{worst}$ and APH. The $H3_{best}$ decreases the average bucket size by at most '9%' and by at least '1.5%' in comparison to APH hashing function.

The number of overflow items (these represent the items that can not be placed in the related bucket with specified size) in different hashing functions with two different bucket size, is presented in Tables VI and VII.

Based on the Tables VI and VII, we can observe that the $H3_{best}$ decreases the number of overflow items compared to APH and $H3_{worst}$. From the Table V, we can observe that, the average bucket size for all of hashing functions is less than '2'. But the Table VI and Table VII show that the size of some buckets is larger than '3'. It is due to the distribution of rules in the different tuples. Our observations show that more than half of rules are stored in two tuples and other tuples only include a small number of rules. In the tuple with small number of rules, the number of collisions are low. In the tuples with large number of rules, the number of collisions is more than other tuples and the average size of buckets is larger. Finally, utilizing $H3_{best}$ decreases the average bucket size in the tuple with large number of rules in comparison to other tuples. From Tables VI and VII, we can observe that the $H3_{best}$ with a load factor of $\frac{n}{2^{[log_2^n]+1}}$ generates the lowest number of collisions, the shortest bucket size and the lowest number of overflows in comparison to other hashing functions.

V. OVERALL CONCLUSIONS

In this paper, we presented an introduction to the classic packet classification problem and one classification algorithm called tuple search. An overview of hashing functions as part of packet classification algorithm in the tuple space was given. We implemented an adaptive software packet classifier that finds optimal hashing function through the class $H3$ hashing functions which is called $H3_{best}$.

Our implementation shows that the utilization of an $H3$ class hashing functions in adaptive packet classification system decreases the number of collisions and average bucket size in comparison to other hashing functions. Deduction in number of collisions, increases packet classification performance and throughput.

REFERENCES

[1] M. Ahmadi and S. Wong. "Modified Collision Packet Classification Using Counting Bloom Filter in Tuple Space". In *Procedings of the 25'th IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN 2007)*, pages 70–76, February 2007.

[2] J. Lawrence Carter and Mark N. Wegman. "Universal Classes of Hash Functions". In *Proceedings of the 9th annual ACM symposium on Theory of computing*, pages 106–112. ACM Press, 1977.

[3] P. Gupta and N. McKeown. "Algorithms for Packet Classification". *Journal of IEEE Network*, 15(2):24–32, March-April 2001.

[4] T. V. Lakshman and D. Stiliadis. "High-speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching". In *Proceedings of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (ACM/SIGCOM)*, pages 203–214, September 1998.

[5] A. Partow. "General Purpose Hash Function Algorithms". http:// www.partow.net/ programming/ hashfunctions/index.html.

[6] M. V. Ramakrishna, E. Fu, and E. Bahcekapili. "Efficient Hardware Hashing Functions for High Performance Computers". *IEEE Transaction Computer*, 46(12):1378–1381, 1997.

[7] H. Song. "Evaluation of Packet Classification Algorithms". http:// www.arl.wustl.edu/ hs1/PClassEval.html, 2006.

[8] H. Song, J. Turner, S. Dharmapurikar, and J. Lockwood. "Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing". In *Proceedings of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 181–192, August 2005.

[9] V. Srinivasan. *"IP Lookup and Packet Classification"*. PhD thesis, Washnigton University, Saint lous, Missouri, August 1999.

[10] V. Srinivasan. "A Packet Classification and Filter Management System". In *Proceedings of International IEEE Conference INFOCOM*, pages 1464–1473, 2001.

[11] V. Srinivasan, S. Suri, and G. Varghese. "Packet Classification using Tuple Space Search". In *Proceeding of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 135–146, 1999.

[12] D. E. Taylor. *"Models, Algorithms, and Architectures for Scalable Packet Classification"*. PhD thesis, Department of Computer Science and Engineering Washington University, August 2004.

Rules-set	**Function**	**APH**	**DJBH**	**DEK**	**ELF**	**JS**	**CRC**
FW1-1k		282	308	301	301	290	299
FW1-5k		1677	1688	1701	1698	1691	1697
FW1-10k		3347	3439	3403	3341	3419	3347

TABLE III

DIFFERENT GENERAL HASHING FUNCTIONS SPECIFICATION.

Rule-set database	$H3_{best}$		$H3_{worst}$		**APH**	
	Load factor		**Load factor**		**Load factor**	
	$\frac{n}{prime(n)}$	$\frac{n}{2^{[log_2^n]+1}}$	$\frac{n}{prime(n)}$	$\frac{n}{2^{[log_2^n]+1}}$	$\frac{n}{prime(n)}$	$\frac{n}{2^{[log_2^n]+1}}$
FW1-100	19 (-%13)	8(-%66)	40	33	22	24
FW1-1k	273 (-%3)	161(-%17)	350	246	282	195
FW1-5k	1664(-%1)	777(%6)	1822	935	1677	824
FW1-10k	3242(-%3)	1615(-%6)	3633	1839	3347	1727
FW1	73 (-%14)	**18(-%49)**	116	63	84	35
ACL1	**235(-%7)**	109(-%17)	323	203	252	131
IPC1	**469(-%7)**	207(-%28)	599	349	504	286

TABLE IV

THE NUMBER OF COLLISIONS FOR DIFFERENT HASHING FUNCTIONS WITH DIFFERENT LOAD FACTORS.

Rule-set database	$H3_{best}$		$H3_{worst}$		**APH**	
	Load factor		**Load factor**		**Load factor**	
	$\frac{n}{prime(n)}$	$\frac{n}{2^{[log_2^n]+1}}$	$\frac{n}{prime(n)}$	$\frac{n}{2^{[log_2^n]+1}}$	$\frac{n}{prime(n)}$	$\frac{n}{2^{[log_2^n]+1}}$
FW1-100	1.31(-%22)	1.19(-%27)	1.85	1.73	1.34	1.65
FW1-1k	1.57(-%2)	1.40(-%6)	1.84	1.69	1.60	1.50
FW1-5k	1.57(-%2)	1.34(-%2)	1.66	1.42	1.58	1.37
FW1-10k	1.53(-%2)	1.35(-%1.5)	1.65	1.40	1.57	1.37
FW1	1.41(-%7)	1.42(-%7)	1.81	1.58	1.51	1.31
ACL1	**1.49(-%1.5)**	**1.28(-%1.5)**	1.78	1.59	1.51	1.31
IPC1	**1.41(-%1.5)**	**1.26(-%9)**	1.68	1.51	1.49	1.38

TABLE V

THE AVERAGE BUCKET SIZE FOR DIFFERENT HASHING FUNCTIONS AND DIFFERENT LOAD FACTORS.

Rule-set database	$H3_{best}$		$H3_{worst}$		**APH**	
	Load factor		**Load factor**		**Load factor**	
	$\frac{n}{prime(n)}$	$\frac{n}{2^{[log_2^n]+1}}$	$\frac{n}{prime(n)}$	$\frac{n}{2^{[log_2^n]+1}}$	$\frac{n}{prime(n)}$	$\frac{n}{2^{[log_2^n]+1}}$
FW1-100	1	0	20	16	7	8
FW1-1k	72	31	145	100	92	52
FW1-5k	483	140	610	227	446	166
FW1-10k	847	296	1184	419	939	359
FW1	10	0	46	23	19	4
ACL1	56	13	130	62	66	22
IPC1	102	24	209	111	130	57

TABLE VI

THE NUMBER OF OVERFLOWS WHEN THE BUCKET SIZE IS 2.

Rule-set database	$H3_{best}$		$H3_{worst}$		**APH**	
	Load factor		**Load factor**		**Load factor**	
	& $\frac{n}{prime(n)}$	$\frac{n}{2^{[log_2^n]+1}}$	$\frac{n}{prime(n)}$	$\frac{n}{2^{[log_2^n]+1}}$	$\frac{n}{prime(n)}$	$\frac{n}{2^{[log_2^n]+1}}$
FW1-100	0	0	10	6	2	0
FW1-1k	12	0	62	39	27	11
FW1-5k	102	13	189	57	78	26
FW1-10k	163	34	353	102	222	57
FW1	0	0	23	12	5	0
ACL1	6	0	50	18	12	3
IPC1	15	1	77	38	34	7

TABLE VII

THE NUMBER OF OVERFLOWS WHEN THE BUCKET SIZE IS 3.