MODIFIED COLLISION PACKET CLASSIFICATION USING COUNTING BLOOM FILTER IN TUPLE SPACE

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 continues to be an important challenge in network processing. It requires matching each packet against a database of rules and forwarding the packet according to the highest priority matching rule. Within the packet classification hash-based algorithms, an algorithm that is gaining interest is the tuple space search algorithm that groups the rules into a set of tuple spaces according to their prefix lengths. An incoming packet can now be matched to the rules in a group by taking into consideration only those prefixes specified by the tuples. More importantly, matching of an incoming packet can now be performed in parallel over all tuples. Within these tuple spaces, a drawback of utilizing hashing is that certain rules will be mapped to the same location, also called collision. The negative effect of such collision is that it will result in multiple memory accesses and subsequently longer processing time. In this paper, we propose to use a pruned counting Bloom filter to reduce collisions in the tuple space packet classification algorithm. This approach decreases the number of collisions and memory accesses in the rule set hash table in comparison to a traditional hashing system. We propose to utilize the pruned counting Bloom filter to decrease the number of collisions. More specifically, we investigate several well-known hashing functions and determine the number of collisions and show that utilizing the pruned counting Bloom filter the number of collisions can be further reduced by at least 4% and by at most 32% for real rule sets.

KEY WORDS
Packet classification, tuple space, hashing, Bloom filter

1 INTRODUCTION

Traditionally, routers forward packets based on the destination address in the packet. The support of many different services such as Quality of Service (QoS), Virtual Private Network (VPN), policy-based routing, traffic shaping, firewalls, and network security, increases the importance of packet classification. In order to provide these services, the router must categorize the incoming packets according to different criteria. These criteria are determined based on one or more fields in the packet header. Packet header fields include destination and source IP addresses, the protocol type, and the destination and source port numbers as depicted in 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. A packet classifier can be implemented in either software or hardware. 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 thereof) 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 based one the tuple (the number of tuples depends on the rule set) and hashed to determine whether it matches a rule within that particular tuple space. A common problem in using hashing is collision, i.e., the mapping of rules (within a single tuple space) 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 a much longer processing time.

In this paper, we propose to utilize the pruned counting Bloom filter to decrease the number of mentioned collisions. More specifically, we investigate several well-known hashing functions and determine the number of collisions and show that utilizing the pruned counting Bloom filter the number of collisions can be reduced by at least 4% and by at most 32% for real rule sets.

This paper is organized as follows. Section 2 describes a summary of related work in this area. Section 3 describes the tuple space algorithms for packet classification, the definition of Bloom filter, and their implementations. Section 4 presents our results. Section 5 presents the

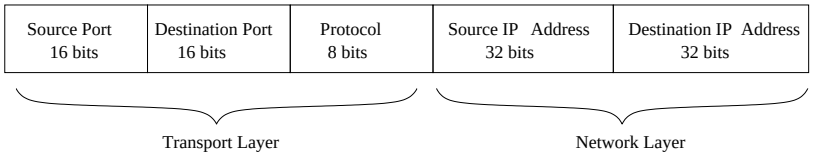

Figure 1. Most important fields that are used in classification algorithms.

overall conclusions.

2 RELATED WORK

The packet classification problem is inherently hard from a theoretical standpoint[1]. It has been shown that the packet classification requires either $O(logN^{k-1})$ time and linear memory, or $log(N)$ time and $O(N^K)$ memory, where N is the number of rules, and K is the number of fields in the header used in the rules[1, 7]. Several research groups proposed a collection of software and hardware solutions [1, 5, 7, 13]. These solutions are: exhaustive search, decision tree, grid-of-Tries, decomposition and tuple space search [5, 13, 12].

Many algorithms exist in the packet classification area and we discuss here only those algorithms that are related to our work. The Bloom filter recently has been used for network applications, one interesting effort has been performed by Darmpurikar and Song who used Bloom filters for packet classification and IP lookup problems [4, 10]. Another work to reduce collisions and access times in the hash table proposes Fast Hash Table (FHT) architecture [10] that convert a Bloom filter into a counting Bloom filter and associate a hash bucket with it. The former work involved a TCAM memory and a crossproducting algorithm that merges the tuple space with a Bloom filter and later effort only show simulation results with many hashing functions and memory buckets per item. In our work, we implement a software packet classifier using a pruned counting Bloom filter in the traditional tuple space. Our approach utilizes a pruned counting Bloom filter to organize the rule set in a tuple space with a minimized number of hashing functions and collisions. We show related results and implement a classifier software tool.

3 TUPLE SPACE AND BLOOM FILTER CONCEPT

In this section, we present the concept of the tuple space classification algorithm and Bloom filters.

3.1 Tuple space classification

A high-level approach for multiple field search employs tuple spaces with a tuple representing information in each field specified by the rules. Srinivasan, et.al. introduced the tuple space approach and the collection of tuple search algorithms in [11, 12]. We provide a simplified example rule

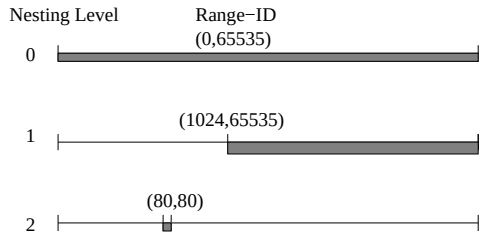

Figure 2. Assigning values for ranges, based on the Nesting Level and the Range-ID.

classification on five fields in Table 1. 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 protocol is specified, $'0'$ if a wildcard is specified [12, 13]. The number of specified bits in a port range are less straightforward to define. The authors introduced the concept of nesting levels and Range-IDs to define the tuple value for port ranges. The nesting level specifies the layer of the hierarchy and the Range-ID uniquely labels the range within its layer. In this manner, all port ranges can be converted to a (Nesting level, Range-ID) pair. We present in the following an example to illustrate Range-IDs. 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 a port range to a nesting level and a Range-ID for Table 1 is depicted in Figure 2.

In the following, we illustrate how a search key is constructed from a packet based on a tuple. 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. Finally, all algorithms using the tuple space approach involve a search of the tuple space or a subset of the tuples in the tuple space.

3.2 Counting Bloom filter

A Bloom filter is a simple space efficient randomized data structure for representing a set in order to support membership queries. Bloom filters allow false positives, but the space savings often outweigh this drawback when the probability of an error is made sufficiently low. Burton Bloom introduced Bloom filters in the 1970s [2].

A set $S = x_1, x_2, ..., x_n$ of n elements is repre-

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)	*	*	[32,32,0,0]
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	[32,32,1,1]
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 1. Simpilified example of rule classification.

sented by an array v of m bits that are initially all set to 0. A set of k independent hash functions, $h_1, h_2, ...h_k$ each with range $1, 2, ..., m$, are used to set the bits at positions $h_1(x), h_2(x), ..., h_k(x)$, for all x in set S. For each element $x \in S$, the bits at position $h_i(x)$ are set to 1 for $1 \leq i \leq k$. A location can be set to 1 multiple times, but only the first change has an effect. To check if an item y is in S, we check whether all $h_i(y)$ are set to 1. If not, then clearly y is not a member of S. If all $h_i(y)$ are set to 1 then y is in S. For example, imagine that enough keys are inserted that all bits in v have been set; in this case, every search will be successful whether or not the key was actually inserted [6]. An example of Bloom filter is depicted in Figure 3.

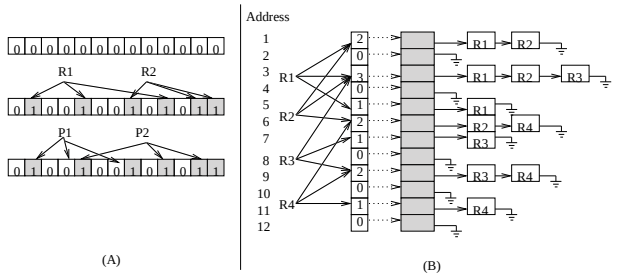

Figure 3. (A) An example of a Bloom filter (B) The hash table structure using counting Bloom filter for four rules.

Figure 3(A) depicts a Bloom filter structure and two rules $R1$ and $R2$, starting with an array containing all '0's. Each item of rule set R_i is hashed k times using k hash function and the related bits are set to 1. To check whether element R_i is in the set hash it k times using the same hash functions and check the corresponding bits. Based on Figure 3(A), element $P1$ cannot be in the set since a zero is found at one of the bits locations and $P2$ can be in the set since all positions contain ones.

A variant of the Bloom filter is the counting Bloom filter [10] in which each bit of the filter is replaced by a counter. The insertion of an item results in incrementing the counter indexed by the corresponding hashing function. Therefore, a counter in this filter essentially represent the number of items hashed to its location. Figure 3(B) depicts hashing four rules to a hash table based on a counting Bloom filter. The counter of the array show the number of elements in the related bucket. We compute k hash functions $h_1(), ..., h_k()$ over an input item and increment the related k counters. Subsequently, we store the item in the lists associated with each of k buckets hence a single item is stored k times in memory. In this approach, we need to maintain up to k copies of each item requiring k times more memory compared to a traditional hash table. However, in a Bloom hash table only one copy is accessed while the other $(k-1)$ copies of item are never accessed. Therefore, the memory requirement can be minimized in the mentioned structure, resulting in the pruned counting Bloom hash table. The pruned Bloom hash table for Figure 3(B) is depicted in Figure 4.

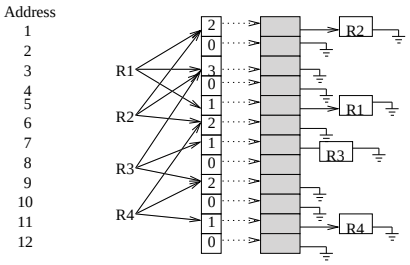

Figure 4. The hash table structure using pruned counting Bloom filter.

An alternative to the pruned counting Bloom filter, insertion and deletion of redundant items are preformed simultaneously. In this method, the memory requirement is equal to n, i.e., the number of elements in the hash table. In the meantime, it involves more computations in the hash table construction time relative to the previous method. It must be noted that during the pruned Bloom filter hash table creation, the counter values are not changed thereby, after pruning, the counter does not express the number of items in the list and is greater than or equal to the number of items in each bucket[10].

3.3 Implementation

In this section, we present the architecture of a packet classifier and the resulting Bloom filter implementation in tuple space. For testing the system, we use different rule set databases and packet traces that have been used by the Applied Research Laboratory in Washington University in St. Louis [9]. The specification of rule databases and packet traces is shown in Table 2. Table 2 includes seven rule sets database and packet traces. The rule sets FW1, ACL1, IPC1are extracted from real rule sets and other generated by the classbench benchmark. More details on classbench, rule sets database, and packet traces can be found in [9].

3.3.1 The packet classifier architecture

In this paper, we merge the pruned Bloom filter and tuple space for packet classification to decrease the number of collisions and memory accesses in rules hash table. The

two main parts of our architecture are: rules hash table constructor and packet search. The architecture is depicted in Figure 5.

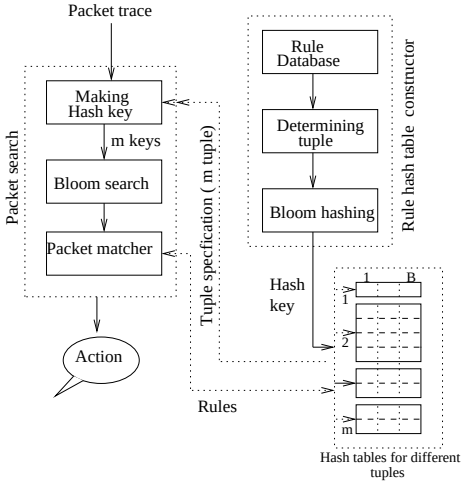

Figure 5. The architecture of classifier using pruned Bloom filter and tuple space.

Rule hash table constructor: This component reads the rules from a rule set database and extracts the rule specification to determine the related tuple that the rule belongs to. After determining the tuple, the rule should be hashed, the next procedure is making a hash key using the Bloom filter and finally store the rules in the hash table. In this process, different hash tables with unequal size are created, since each hash table correspond to a single tuple and each tuple included different rules and tuple specifications. Usually more than half of the rules belong to two tuples, and this is depicted on the right side of Figure 5. In this figure, m is the number of tuples and B shows the size of the buckets.

Packet search: This component of the classifier processes the incoming packets to find matching rules in the hash tables corresponding to tuples. Therefore, for each incoming packet, a hash key is extracted based on each tuple specification. Subsequently, m hash keys are used to access the m hash tables (after hashing) to determine whether matching rules can be found. 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. 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, thus the number of access in sequential search process per packet is equal to the number of tuples.

4 RESULTS

In this section, we present the results of several experiments utilizing the proposed architecture for different rule set databases. In the classifier architecture, an important element entails the hashing functions that are used by the Bloom filter. We utilize seven hashing function in this architecture [8]. The specification of hashing functions are presented in Table 2. The table includes the name, the minimum required clock cycle and the number of generated collisions for different rule set databases. In Table 3, we can observe that four hashing functions with acceptable execution time and collision are (DJB, SDBM, JS, ELF) and (DJB, SDBM, APH, DEK), respectively. The hashing function execution time is measured as the number of clock cycles that the function runs. The number of clock cycles in Table 3 have been obtained after consecutive runs and then selecting the minimum number of clock cycles for each function. To provide greater precision for timing measurements, many processors also contain a timer that operates at the clock cycle level[3]. We compute the number of collisions for different bucket sizes and several hashing functions including two groups of four hash functions in the Bloom filter. The results are presented in Tables 4 and 5. In Tables 4 and 5, Bloom-k denotes a Bloom filter with k hashing functions. The data of each column with the bucket size set to 1 shows the number of collisions and the remaining columns show the number of overflow items. Based on Tables 4 and 5, we can observe that using a pruned counting Bloom filter decreases the number of collisions when the number of hashing functions is 2, in this case number of collisions reduced by 32%. For real rule sets, the number of collisions is reduced by at least 4% for Bloom-4 and rule set IPC1 and by 32% for Bloom-2 and rule set IPC1. The worst case involves Bloom-4 and FW-100. In this case, the number of tuples is 26 and number of rules is 92 with more than 50% of rules grouped in two tuples. Consequently, the other tuples encompass a low number of rules. In other words, the rate of randomization by the Bloom filter is decreased when a tuple has a low number of rules. In Tables 4 and 5, we can observe that the Bloom-2 results in the lowest number of collisions. The decreasing number of collisions causes a decrease in the number of accesses per packet. Additionally, we calculated the average number of accesses per packet for different rules databases and packet traces. These results are presented in Table 6. In general, hashing systems create collisions. In here, we use overflow area and chaining techniques for collision solving. Utilizing an overflow area, i.e, a shared memory area storing all colliding rules, requires again a linear search through this area, and is therefore no longer considered. In the chaining technique, each colliding rule is stored in a dynamic memory cell that is pointed to by the related bucket. In the following, we only present the results of the chaining technique (see Table 6). We can observe in Table 6 that the number of accesses per packet converge to the number of tuples in the rule sets database. In the pruned counting Bloom filter, the number of accesses per packet is minimized when the number of functions in the pruned counting Bloom filter is 2. In this case for rule sets databases FW1-1k, FW1-5k, FW1-10k and FW1, the number of accesses per packet using Bloom-2 decreases by 19% in comparison to the traditional hashing system. Note that the results have been generated for rule set with corresponding packet trace. In

Rule 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
Packet trace	**FW1-100**	**FW1-1k**	**FW1-5k**	**FW1-10k**	**FW1**	**ACL1**	**IPC1**
Number of Packets	920	8050	46700	93250	2830	8140	17020

Table 2. Rule set database and packet trace specification.

Function Rules	APH	DJBH	SDBM	DEK	ELF	JS	CRC	One-at-time
FW1-1k	288	308	290	301	301	290	299	288
FW1-5k	1669	1688	1645	1701	1698	1691	1697	1709
FW1-10k	3424	3439	3357	3403	3341	3419	3347	3411
Minimum clock cycle	824	396	460	492	612	448	2864	867

Table 3. Different hashing functions specification used in pruned counting Bloom implementation.

	Bloom-4			Bloom-3			Bloom-2			Traditional hashing		
	Bucket Size			Bucket Size			Bucket Size			Bucket Size		
Rule Set	1	2	3	1	2	3	1	2	3	1	2	3
FW1-100	28 (+27%)	4	0	21 (-5%)	2	0	19 (-14%)	1	0	22	6	2
FW1-1k	271 (-7%)	49	3	226 (-23%)	25	0	202 (-31%)	27	0	292	101	27
FW1-5k	1615 (-2%)	215	5	1353 (-18%)	127	5	1211 (-27%)	96	4	1651	475	112
FW1-10k	3260 (-1%)	480	13	2710 (-18%)	229	3	2373 (-28%)	177	3	3305	912	209
FW1	76 (-5%)	8	0	64 (-20%)	4	0	58 (-28%)	3	0	80	19	7
ACL1	254 (-6%)	40	1	209 (-23%)	19	0	187 (-31%)	15	0	271	76	19
IPC1	476 (-7.5%)	61	4	402 (-22%)	36	2	354 (-31%)	28	1	515	148	40

Table 4. The number of collisions and overflows in pruned counting Bloom filter with DJB, SDBM, APH, DEK hashing functions compared with traditional hashing system.

	Bloom-4			Bloom-3			Bloom-2			Traditional hashing		
	Bucket Size			Bucket Size			Bucket Size			Bucket Size		
Rule Set	1	2	3	1	2	3	1	2	3	1	2	3
FW1-100	28 (+27%)	3	0	26 (+18%)	2	0	22 (0%)	3	0	22	7	2
FW1-1k	271 (-8%)	44	2	217 (-26%)	26	0	203, (-31%)	24	2	294	95	28
FW1-5k	1635 (-4%)	237	2	1358 (-20%)	129	5	1188 (-30%)	78	4	1705	454	80
FW1-10k	3287 (-1.4%)	499	14	2710 (-19%)	247	2	2396 (-28%)	195	3	3335	921	211
FW1	76 (-5%)	10	0	64 (-20%)	5	0	58 (-28%)	3	0	80	19	7
ACL1	253 (-7%)	39	2	207 (-24 %)	22	1	187 (-31%)	15	0	271	76	19
IPC1	495 (-4%)	63	2	413 (-20%)	27	0	349 (-32%)	28	1	515	148	40

Table 5. The number of collisions and overflows in pruned counting Bloom filter with DJB, SDBM, JS, ELF hashing functions compared with traditional hashing system.

	Bloom-4			Bloom-3			Bloom-2			Traditional hashing		
	Bucket Size			Bucket Size			Bucket Size			Bucket Size		
Rule Set & packet trace	1	2	3	1	2	3	1	2	3	1	2	3
FW1-100	26.5	26	26	26.6	26	26	26	26	26	26.8	26	26
FW1-1k	44.4	42	42	42	42	42	42	42	42	47.5	43	42
FW1-5k	55	52	52	54	52	52	52	52	52	64	59.6	58
FW1-10k	58.5	55	55	58	55	55	55	55	55	68	62.5	61
FW1	36	36	36	36	36	36	36	36	36	44.6	43	42
ACL1	44.5	44	44	44.2	44	44	44	44	44	45.6	44.2	44
IPC1	182	179	179	180	179	179	180	179.5	179	188	181	179.6

Table 6. The average number of access per packet for related packet trace using chaining in pruned counting Bloom filter compared with traditional hashing system.

	Bloom-4				Bloom-3				Bloom-2				Traditional hashing			
	Bucket Size				Bucket Size				Bucket Size				Bucket Size			
Rule Set	1		2		1		2		1		2		1		2	
Action time	H	M	H	M	H	M	H	M	H	M	H	M	H	M	H	M
FW1-100	520	729	515	385	467	666	463	396	451	533	449	389	492	559	491	399
FW1-1k	547	1016	547	568	481	678	482	396	472	487	487	362	532	741	539	531
FW1-5k	558	985	548	542	497	686	495	421	485	585	490	389	562	1394	568	1107
FW1-10k	565	1146	569	471	515	467	505	412	493	655	497	446	569	1438	575	1098
FW1	526	935	523	403	471	657	463	379	468	461	456	321	508	1517	515	1229
ACL1	545	636	529	304	471	285	463	279	443	330	451	246	499	477	498	342
IPC1	549	835	552	406	500	570	485	338	475	431	431	311	563	583	557	405

Table 7. The average number of clock cycle per packet in each tuple for different rule sets and packet trace using chaining.

our implementation, we have implemented our architecture sequentially. In other words, the hash key calculation in the Bloom filter and traditional hashing computed by sequential code. Hence, we expect the sequential search time for Bloom filter to be more than traditional hashing. The results of searching packet time within tuple are presented in Table 7. In this table, each column includes two other sub-column with labels H an M, the H column represent the hashing function execution time and the M column represent the memory access time both in clock cycles. Based on Table 7, we can observe that the growth of the number of hashing functions in the Bloom filter increases the hashing time and decreases memory access time. The best results are generated when the number of hashing functions in Bloom filter are 2. In this case the number of required clock cycles decline for all rule sets.

5 OVERALL CONCLUSIONS

In this paper, we presented an introduction to a packet classification problem and one classification algorithm, i.e., tuple search. Subsequently, we presented the Bloom filter and a modified version of the Bloom filter as part of a packet classification algorithm using the tuple space. We implemented the Bloom filter using a class of hashing functions and traditional hashing system as part of software packet classifier.

Our implementation shows that using a pruned counting Bloom filter in a packet classification system can decrease the number of collisions and accesses per packet relative to a traditional hashing system. In the best case for two hashing functions, the number of collisions and memory accesses was reduced by 32% and 19%, respectively.

References

[1] F. Baboescu, S. Singh, and G. Varghese. Packet Classification for Core Routers: Is There an Alternative to CAMs? In *Proc. 22'th Int'l Conf. IEEE INFOCOM*, pages 53–63, March-April 2003.

[2] B. H. Bloom. Space /Time Trade-offs in Hash Coding with Allowable Errors. *J. Communication of the ACM*, 13(7):422–426, July 1970.

[3] R. E. Bryant and D. R. O'Hallaron. *Computer Systems: A Programmer's Perspective (Hardcover)*. Prentice Hall, 1st edition, August 2002.

[4] S. Dharmapurikar, H. Song, J. Turner, and J. Lockwood. Fast Packet Classification Using Bloom Filters. Technical Report 27, Department of Computer Science And Engineering, Washington University in St. Louis, May 2006.

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

[6] S. Kumar and P. Crowley. Segmented Hash: An Efficient Hash Table Implementation for High Performance Networking Subsystems. In *Proc. Symp. on Architecture for Networking and Communications Systems (ANCS05)*, pages 91–103, October 2005.

[7] T. V. Lakshman and D. Stiliadis. High-speed Policy-based Packet Forwarding using Efficient Multi-dimensional Range Matching. In *Proc. Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication (ACM/SIGCOM)*, pages 203–214, 1998.

[8] A. Partow. General Purpose Hash Function Algorithms. http:// www.partow.net/ programming/ hash-functions/index.html.

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

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

[11] V. Srinivasan. A Packet Classification and Filter Management System. In *Proc. Int'l IEEE Conf. INFOCOM*, pages 1464–1473, 2001.

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

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