Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2011, Article ID 475865, 10 pages
doi:10.1155/2011/475865

Research Article

A Cache Architecture for Counting Bloom Filters: Theory and Application

Mahmood Ahmadi[1, 2, 3] and Stephan Wong[1]

[1] *Computer Engineering Laboratory, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, 2600 AA Delft, The Netherlands*
[2] *Department of Computer Engineering, Faculty of Engineering, University of Razi, Kermanshah, Iran*
[3] *Computer Engineering Department, Scince and Research Branch, Islamic Azad University of Kermanshah, Kermanshah, Iran*

Correspondence should be addressed to Mahmood Ahmadi, m.ahmadi@razi.ac.ir

Received 7 March 2011; Revised 19 June 2011; Accepted 24 June 2011

Academic Editor: Parag K. Lala

Within packet processing systems, lengthy memory accesses greatly reduce performance. To overcome this limitation, network processors utilize many different techniques, for example, utilizing multilevel memory hierarchies, special hardware architectures, and hardware threading. In this paper, we introduce a multilevel memory architecture for counting Bloom filters. Based on the probabilities of incrementing of the counters in the counting Bloom filter, a multi-level cache architecture called the cached counting Bloom filter (CCBF) is presented, where each cache level stores the items with the same counters. To test the CCBF architecture, we implement a software packet classifier that utilizes basic tuple space search using a 3-level CCBF. The results of mathematical analysis and implementation of the CCBF for packet classification show that the proposed cache architecture decreases the number of memory accesses when compared to a standard Bloom filter. Based on the mathematical analysis of CCBF, the number of accesses is decreased by at least 53%. The implementation results of the software packet classifier are at most 7.8% (3.5% in average) less than corresponding mathematical analysis results. This difference is due to some parameters in the packet classification application such as number of tuples, distribution of rules through the tuples, and utilized hashing functions.

1. Introduction

Most network devices, for example, routers and firewalls, need to process incoming packets (e.g., classification and forwarding) at wire speeds. These devices mostly incorporate special network processors that are comprised of a programmable processor core with several memory interfaces and special coprocessors that are optimized for packet processing. The performance of these network processors is usually hampered by slow (main) memory accesses. Such memory bottlenecks can be overcome by the following mechanisms: hiding of memory latencies through parallel processing and reducing the memory latencies by introducing a multi-level memory hierarchy incorporating special-purpose caches [1]. A poorly designed cache memory can critically affect the performance of network processor since the number of memory accesses required for each lookup can vary. Therefore, high-throughput applications require search techniques with more predictable worst-case lookup performance. An approach to achieve higher lookup performance is to utilize the Bloom filter that is recently utilized in embedded memories [2–4]. A Bloom filter is a simple space-efficient randomized data structure to represent a set in order to support membership queries [5]. There are numerous networking problems where such a data structure is required. In particular, when space is an issue, a Bloom filter may be an excellent alternative to keep an explicit list. A Bloom filter is frequently utilized in network processing (areas), such as packet classification, packet inspection, forwarding, p2p networks, and distributed web caching [5–7]. Therefore, Bloom filters are useful to design high-performance memory architecture in network processors and algorithmic solution in the network processing applications.

In this paper, we introduce a new multi-level cache architecture called the *cached counting Bloom filter* (CCBF). In the CCBF, the cache levels are defined based on the value of the counters. In other words, the items with same counter values are stored in the same level. Based on the counting Bloom filter (CBF) analysis, we propose two multi-level cache architectures (an *l-level* and a 3-*level* one) and, subsequently, present the performance analysis. The performance metric is the number of accesses to different cache levels of the CCBF as compared to the standard Bloom filter. In the 3-level cache, we further determine the size of cache levels for optimal false positive probabilities. To test the CCBF, we implemented a software packet classifier utilizing a 3-level CCBF employing tuple spaces that are traditionally utilized in packet classification. The mathematical analysis and software implementation results show that the number of accesses is decreased when a 3-level CCBF is utilized. Based on the mathematical analysis, the number of accesses is decreased by at least 53% in comparison to the standard Bloom filter. The results of software implementation have at most 7.8% (3.5% in average) difference to corresponding results of mathematical analysis. This difference is due to the following reasons: number of tuples, distribution of rules inside the tuples, and utilized hashing functions. The main contributions of this paper are the following:

(i) introducing of Bloom filter variant called *cached counting Bloom filter* (CCBF)

(ii) mathematical analysis of the CCBF

(iii) the performance evaluation of the proposed CCBF for packet classification.

The rest of paper is organized as follows. Section 2.2 presents related works. Section CCBF describes a cache counting Bloom filter concept, architecture, and analysis. The case study of packet classification is presented in Section 3. Section 4 presents our analysis and software packet classifier implementation results. In Section 5, we draw the overall conclusions.

2. Related Works

In this section, we take a brief look at previous works regarding the packet classification using Bloom filter and memory organization in Bloom filters. In [8, 9], Srinivasan et al. introduced the tuple space approach and the collection of tuple search algorithms. 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. The tuple-based algorithms utilize traditional hashing system. In [4], an extended version of the Bloom filter was considered. The authors presented a fast hash table architecture (FHT) and lookup algorithm that converts a Bloom filter into a counting Bloom filter and an associated hash bucket. The FHT improves the performance over a standard hash table by reducing the number of memory accesses needed for the most time-consuming lookups. It only works in conjunction with counting Bloom filters and needs to reconsider all of the already inserted items for each item that consequently leads to longer processing time. In [10], a hash architecture called a multi-predicate Bloom-filtered hash table (MBHT) using parallel Bloom filters is presented. It generates off-chip memory addresses in the base-2^x number system, $x \in \{1, 2, \ldots\}$, which removes the overhead of pointers. Using a larger base of number system, an MBHT reduces on-chip memory size. In [11], an approach to packet classification which combines architectural and algorithmic techniques is presented. The starting point is the well-known crossproduct algorithm which is fast but has significant memory overhead due to additional rules needed to represent the crossproducts. The proposed approach modifies the crossproduct method to reduce the memory requirement. Unnecessary accesses to the off-chip memory are avoided by filtering them through on-chip Bloom filters. In [7], a cache design based on the standard Bloom filter was investigated and was extended to support ageing (adding the ability to evict stale entries from the cache), bound misclassification rates, and use multiple binary predicates. It examined the exact relationship between the size and dimension of the number of flows that can be supported and the misclassification probability incurred. Additionally, it presented extensions for gracefully ageing the cache over time to minimize misclassification. In [12], we introduce the concept of CCBF and proposed two architectures for the counting Bloom filters and their mathematical analysis. In this paper, we implement a CCBF in packet classification using tuple space search with a class $H3$ of universal hashing functions. Consequently, we compare the software implementation and mathematical analysis results of the CCBF to a standard Bloom filter. The experimental results show that the utilization of the CCBF increases the performance of counting Bloom filters.

2.1. Counting Bloom Filter. The standard Bloom filter works fine when the members of the set do not change over time. When they do, adding items requires little effort since it only requires hashing the additional item and setting the corresponding bit locations in the array. On the other hand, removing an item conceptually requires unsetting the ones in the array, but this could inadvertently lead to removing a 1 that was the result of hashing another item that is still member of the set. To overcome this problem, the counting Bloom filter (CBF) was introduced [13]. In the counting Bloom filter, each bit in the array is replaced by a small counter. When inserting an item, each counter indexed by the corresponding hash value is incremented; therefore, a counter in this filter essentially represents the number of items hashed to it. When an item is deleted, the corresponding counters are decremented. In the following, we utilize $c(i)$ to denote the counter value associated with each i'th counter. Considering a counting Bloom filter for n items, with k hashing functions, and m counters, the probability that the i'th counter is incremented j times is given as a binomial random variable in the following:

$$p(c(i) = j) = \binom{nk}{j} \left(\frac{1}{m}\right)^{j} \left(1 - \frac{1}{m}\right)^{nk-j}. \quad (1)$$

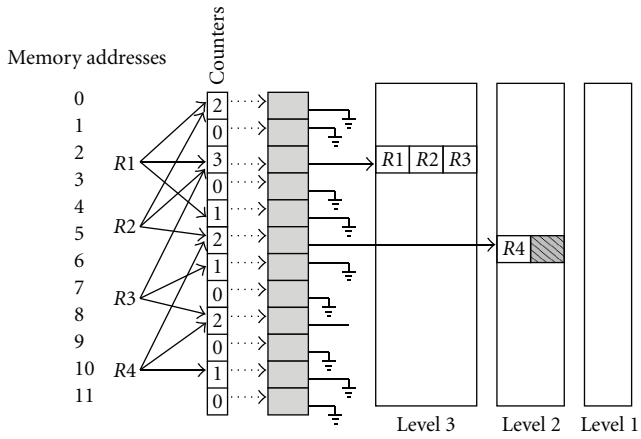

Figure 1: The hash table architecture using cached counting Bloom filters.

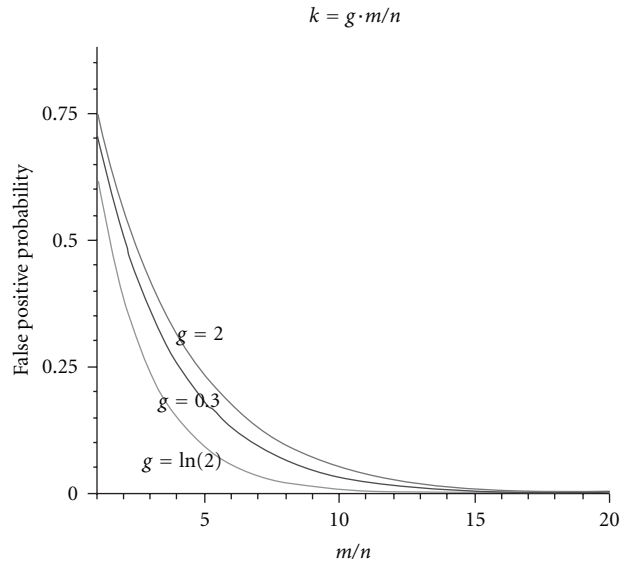

Figure 2: False positive probability for different configurations.

When using n-bit counters, an n-bit counter will overflow if and only if it reaches a value of 2^n. The analysis performed by Fan et al. [13] shows that a 4-bit counter is adequate for most applications.

2.2. Cached Counting Bloom Filter Concept. A cache counting Bloom filter (CCBF) is a counting Bloom filter with multi-level hash table in which the items with the same counter value are stored in the same memory (cache) level. If the number of levels is l, the multi-level counting Bloom filter is called l-level CCBF. We show that in practice, the 3-level CCBF is more beneficial than l-level CCBF. In the CCBF, two types of operations are defined. First type of operations is related to the programming and querying of the Bloom filter, and second type is insertion/deletion and fetching of an item from multi-level CCBF based on the counter values in the counting Bloom filter. This means that in the programming of the Bloom filter, the items are inserted in the related cache level of the CCBF. In querying step, the counter values are checked, and the items from the related cache level are loaded. The operations in cache levels are similar to the operations in the traditional hash table (insertion/deletion and fetching). An example of a Bloom filter and corresponding CCBF is depicted in Figure 1.

Figure 1 depicts the CCBF. To generate the CCBF, each item with its counter is inspected after the Bloom filter is created. The item with maximum counter value is selected to write on the cache level. In this case, item $R1$ is hashed to addresses 1, 3, and 5. Address 3 has maximum value of counters; therefore, $R1$ is stored in level 3. Item $R2$ is hashed to addresses 1, 3, and 6 with the counter values 2, 3 and 2. Therefore, item $R2$ is stored in address 3 in level 3. Similarly $R4$ and $R5$ are stored in level 3 and level 2, respectively. From Figure 1, it can be observed that items $R1$, $R2$, and $R3$ are stored in a bucket in the third level, and item $R4$ is stored in level 2 because of its counter value. In other words, only 2 accesses are required in total, one for the bucket in the third level and the other for the bucket in the second level. It should be noted that each address in the array of counters points to one level. The addresses with the counter value more than 2 points to level 3, the addresses with the counter value 2 points to level 2, and the addresses with the value 1 points to level 1.

According to definition of a Bloom filter, the number of hashing functions (k) with m counters and n items can be expressed as follows [12]:

$$k = g\frac{m}{n}, \tag{2}$$

where the value of g changes for different Bloom filter configurations. Based on the Bloom filter definition, the optimal value for g to have a minimum false positive rate is $g = \ln(2)$ (see Figure 2).

After substituting (2) in (1), we obtain

$$p(c(i) = j) = \binom{gm}{j}\left(\frac{1}{m}\right)^{j}\left(1 - \frac{1}{m}\right)^{mg-j}. \tag{3}$$

Using (3), we can compute the probability of incrementing the i'th counter for different values of g and m. Using (3), the counter probability distribution for different counting Bloom filter configurations is depicted in Figure 3.

From Figure 3, when $g \leq \ln(2)$, the value of the counters with nonzero probability changes between 0 and 3, and when $g > \ln(2)$, the value of the counters with non-zero probability is increased (for $g = 2$, the value of the counters changes between 0 and 5). Therefore, we can utilize a multi-level cache memory to store the items. We introduce the cached counting Bloom filter as a Bloom filter with each counter pointing to the level corresponding to its counter value and each entry in level l containing buckets with size l. A bucket is a set of items that can be transferred in one I/O operation. Therefore, for a Bloom filter with optimal false probability, we can utilize a multi-level caching memory to store the items. The l-level cached counting Bloom filter architecture is depicted in Figure 4.

In this figure, $C(i_l, l)$ represents the counter with the value "l" pointing to location i_l within cache level "l". Therefore, the values of $C_{1,1}, \ldots, C_{i_1,1}$ are equal to 1, the values of $C_{1,l-1}, \ldots, C_{i_{l-1},l-1}$ are equal to $l-1$, and the values of $C_{1,l}, \ldots, C_{i,l}$ are equal to l. The counters with value 0 do not point to any bucket in the cache memory.

2.3. Counting Cached Bloom Filter Analysis. In this section, we present the analysis of the cached counting Bloom filter. The number of accesses to the memory depends on the fact that the Bloom generates a "positive" or "negative" result. For the negative case, no accesses to the memory are needed since it is certain that they are not in the original set. For the positive case, still it must be verified whether the item in question is a member or not (false positive). Consequently, we assume in the analysis that all tests are on different elements which would result in the testing of n elements (the same number of items in the original set). The number of accesses in a standard Bloom filter is $nk(1+p_f)$ memory accesses, where n represents the number of items, k represents the number of hashing functions, and p_f is false positive probability. The l-level cached counting Bloom filter is depicted in Figure 4. From Figure 4, the number of accesses in l-level CCBF is equal to summation of accesses in all levels as follows:

$$\begin{aligned}&\text{number of accesses in } l\text{-level CCBF}\\ &= (N_1 + \cdots + N_i + \cdots + N_l).\end{aligned} \tag{4}$$

In this equation, N_i represents the number of accesses in level i. Based on definition of the CCBF, the size of a bucket in level i is equal to i. Therefore, in each access, i items can be transferred. Consequently, the number of accesses depended on the number of levels that means the utilization of multi-level cached counting Bloom filter decreases the number of accesses. The number of accesses in level i is equal to the number of buckets in this level. To calculate the number of buckets, the size of level i is divided by size of the bucket in this level. From (1) and (4), the expected number of accesses in CCBF is extended as follows:

$$\begin{aligned}&\text{number of accesses in } l\text{-level CCBF}\\ &= A\left(p(j=1) + \frac{p(j=2)}{2} + \cdots + \frac{p(j=l)}{l}\right)\\ &= A\binom{A}{1}\left(\frac{1}{m}\right)\left(1-\frac{1}{m}\right)^{A-1} + \cdots\\ &\quad + A\binom{A}{l}\left(\frac{1}{l}\right)\left(\frac{1}{m}\right)^{l}\left(1-\frac{1}{m}\right)^{A-l}\\ &\qquad \left(\text{with } A = nk\left(1+p_f\right)\right).\end{aligned} \tag{5}$$

In (5), $p(j=l)$ shows the probability that a counter incitements l times. $A(p(j=l)/l)$ represents the number of expected accesses in level l for a counting Bloom filter with n items and k hashing functions. We can rewrite (5) as follows:

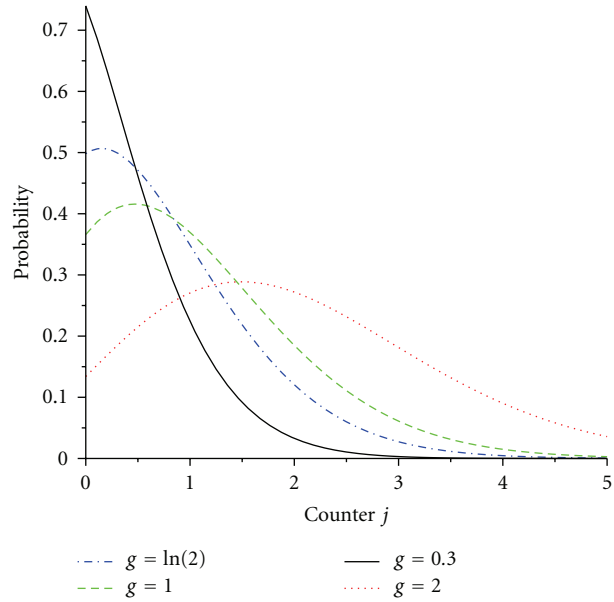

FIGURE 3: The counter probability distribution for different configurations in counting Bloom filters.

$$\begin{aligned}&\text{number of accesses in } l\text{-level CCBF}\\ &= A\left(p(j=1) + \frac{p(j=2)}{2} + \cdots + \frac{p(j=l)}{l}\right)\\ &= Ae^{-A/m}\left(\left(\frac{1}{m}\right)\binom{A}{1} + \cdots + \binom{A}{l}\left(\frac{1}{l}\right)\left(\frac{1}{m}\right)^{l}\right)\\ &\cong Ae^{-A/m}\left(\sum_{i=1}^{l}\frac{1}{ii!}\left(\frac{A}{m}\right)^{i}\right)\\ &\qquad \left(\text{with } A = nk\left(1+p_f\right)\right).\end{aligned} \tag{6}$$

If we assume that $m/n = c$ and normalize to $nk(1+p_f)$, then we can rewrite (6) as follows:

$$\begin{aligned}&\text{number of accesses in } l\text{-level CCBF}\\ &= e^{-k(1+p_f)/c}\left(\sum_{i=1}^{l}\frac{1}{ii!}\left(\frac{k\left(1+p_f\right)}{c}\right)^{i}\right).\end{aligned} \tag{7}$$

In practice, the number of levels is limited. The graph depicted in Figure 3 shows that the counter values likely are not larger than 3. Therefore, a 3-level CCBF is more beneficial than l-level CCBF. We propose to limit the number of levels to 3. More precisely, levels 1 and 2 (containing 1 and 2 buckets, resp.) store the elements for the counters with values 1 and 2, respectively. Level 3 stores the elements for counters with value 3 or larger. As the counters with values larger than 3 require more storage, the elements are

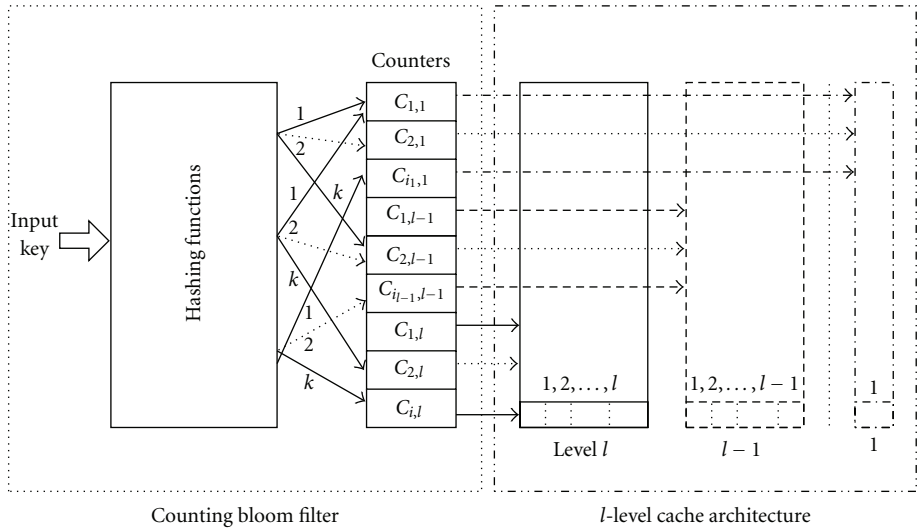

Figure 4: The l-level cached counting Bloom filter architecture.

stored over multiple rows in the third level of the CCBF (*segmentation*). The 3-level cache architecture is depicted in Figure 5.

In Figure 5, the values of $C_{1,1}, \ldots, C_{i_1,1}$ are equal to 1, the values of $C_{1,2}, \ldots, C_{i_2,2}$ are equal to 2, and the values of $C_{1,3}, \ldots, C_{i_3,3}$ are equal to 3. C_{other} represents the counters with values larger than three and, therefore, they point to a storage within level 3 of the CCBF. Figure 5 highlights the mentioned segmentation. In the following, we analyse the effects of the items with counter values larger than three. The number of accesses in a 3-level CCBF is equal to number of accesses in the levels 1, 2, and 3. The number of accesses in third level of cache can be computed as a summation of the number of counters with value 3 and larger. Therefore, the number of accesses in a 3-level CCBF is as follows:

$$\begin{aligned}
&\text{number of accesses in 3-level CCBF} \\
&= A\left(p(j=1) + \frac{p(j=2)}{2} + \frac{p(j\geq 3)}{3}\right) \\
&= A\left(p(j=1) + \frac{p(j=2)}{2} + \frac{p(j=3)}{3}\right) \\
&\quad + \left\lceil \frac{4}{3} \right\rceil p(j=4)A + \cdots + \left\lceil \frac{l}{3} \right\rceil p(j=l)A \\
&= A\binom{A}{1}\left(\frac{1}{m}\right)\left(1-\frac{1}{m}\right)^{A-1} \\
&\quad + A\binom{A}{2}\left(\frac{1}{2}\right)\left(\frac{1}{m}\right)^{2}\left(1-\frac{1}{m}\right)^{A-2} \\
&\quad + A\binom{A}{3}\left(\frac{1}{3}\right)\left(\frac{1}{m}\right)^{3}\left(1-\frac{1}{m}\right)^{A-3} \\
&\quad + \sum_{i=4}^{l}\left\lceil \frac{i}{3} \right\rceil \binom{A}{i}\left(\frac{1}{m}\right)^{i}\left(1-\frac{1}{m}\right)^{A-i} A \\
&\qquad \left(\text{with } A = nk\left(1+p_f\right)\right).
\end{aligned} \tag{8}$$

Equation (8) is represented as follows:

$$\begin{aligned}
&\text{number of accesses in 3-level CCBF} \\
&\cong Ae^{-A/m}\left(\frac{A}{m} + \frac{1}{2*2!}\left(\frac{A}{m}\right)^{2} + \frac{1}{3*3!}\left(\frac{A}{m}\right)^{3}\right) \\
&\quad + \sum_{i=4}^{l}\left\lceil \frac{i}{3} \right\rceil \frac{1}{i!} e^{-A/m}\left(\frac{A}{m}\right)^{i} A \\
&\qquad \left(\text{with } A = nk\left(1+p_f\right)\right).
\end{aligned} \tag{9}$$

After substitution of m/n with c and normalization to $nk(1+p_f)$, the number of accesses in the 3-level CCBF is written as follows:

$$\begin{aligned}
&\text{number of accesses in 3-level CCBF} \\
&\cong e^{-k(1+p_f)/c}\left(\frac{k\left(1+p_f\right)}{c} + \frac{1}{2*2!}\left(\frac{k\left(1+p_f\right)}{c}\right)^{2}\right. \\
&\qquad \left. + \frac{1}{3*3!}\left(\frac{k\left(1+p_f\right)}{c}\right)^{3}\right) \\
&\quad + \sum_{i=4}^{l}\left\lceil \frac{i}{3} \right\rceil \frac{1}{i!} e^{-k(1+p_f)/c}\left(\frac{k\left(1+p_f\right)}{c}\right)^{i}.
\end{aligned} \tag{10}$$

In the following, we evaluate the size of the different cache levels in the CCBF architecture. In short, the size of

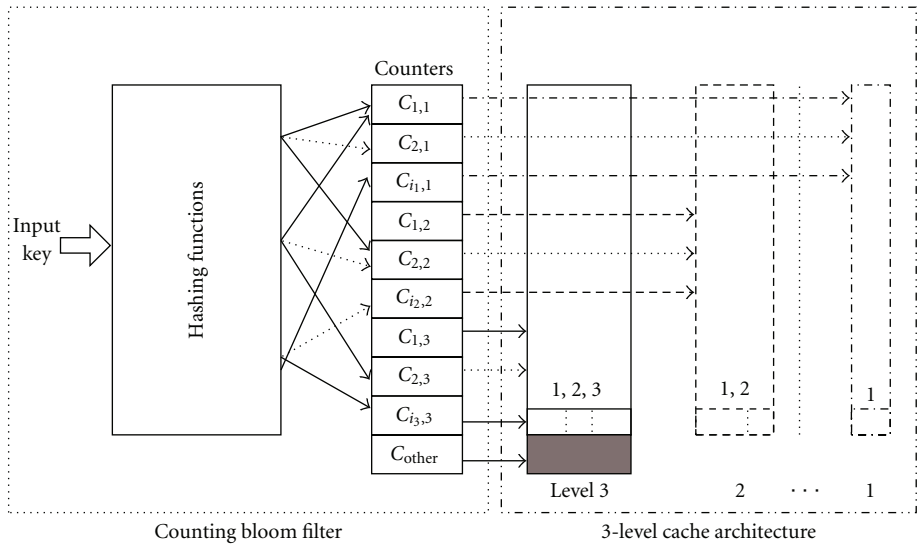

Figure 5: The 3-level cached counting Bloom filter architecture.

each cache level in term of items is equal to the multiplication of nk and the probability of each counter value in the CCBF. The size of each cache level in l-level CCBF is expressed as follows:

$$\begin{aligned}&\text{the size of level } j \text{ within } l\text{-level CCBF}\\ &= nkjp(c(i) = (\text{level number}))\\ &= nkjp(c(i) = j)\\ &= nkj\binom{nk}{j}\left(\frac{1}{m}\right)^{j}\left(1-\frac{1}{m}\right)^{nk-j}.\end{aligned} \tag{11}$$

In (11), j is level number. Using (4), we can rewrite (11) as follows:

$$\text{the size of level } j \text{ in } l\text{-level CCBF} \cong nkje^{-k/c}\left(\frac{1}{j!}\left(\frac{k}{c}\right)^{j}\right), \tag{12}$$

where j is level number.

Using (12), the total size of the l-level CCBF cache after normalization to nk (size of a standard Bloom filter) is

$$\begin{aligned}\text{the total size of } l\text{-level CCBF} &= nkj\sum_{j=1}^{l}(p(c(i) = j))\\ &\cong e^{-k/c}\left(\sum_{j=1}^{l}\frac{j}{j!}\left(\frac{k}{c}\right)^{j}\right).\end{aligned} \tag{13}$$

3. Implementation of a Case Study

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. 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 [14, 15]. For testing purpose, we utilized different rule-set databases and packet traces that have been used by the Applied Research Laboratory in Washington University in St. Louis. The specification of the rule-set databases and packet traces is presented in Table 1.

Table 1 includes seven rule-set databases and packet traces based on IPV4 protocol. The rule sets Fw1, Acl1, and Ipc1 are extracted from real rule sets and others generated by the Classbench benchmark.

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. [8, 9] introduced the tuple space approach and the collection of tuple search algorithms.

A class of universal hashing functions is called H3 hashing functions [16, 17]. Based on tuple space representation for rule-set database and IP packets, the size of input key is 88 bits (32-bit source IP address, 32-bit destination IP address, 8-bit Range-ID, 8-bit Nesting-Level and 8-bit protocol bit). The maximum size of tuple or address space is assumed 2^{16} rules for 16-bit address. Therefore, $Q_{88\times16}$ denotes a set of matrices to define $H3$ hashing function for tuple space packet classification algorithm [18].

4. Performance Evaluation and Results

In this section, we present the mathematical analysis and implementation results of the CCBF architecture in packet classification using tuple space search.

The implementation and mathematical analysis results for Fw1-100, Fw1-1k, and Fw1-5k rule-set databases for a 3-level CCBF are depicted in Figure 6.

Table 1: Rule-set database and packet trace specification.

	Fw1-100	Fw1-1k	Fw1-5k	Fw1-10k	Fw1	Ipc1	Acl1
Rule-set database							
Number of rules	92	971	4653	9311	266	1550	752
Number of tuples	26	42	52	57	36	179	44
Packet trace							
Number of packets	920	8050	46700	93250	2830	17020	8140

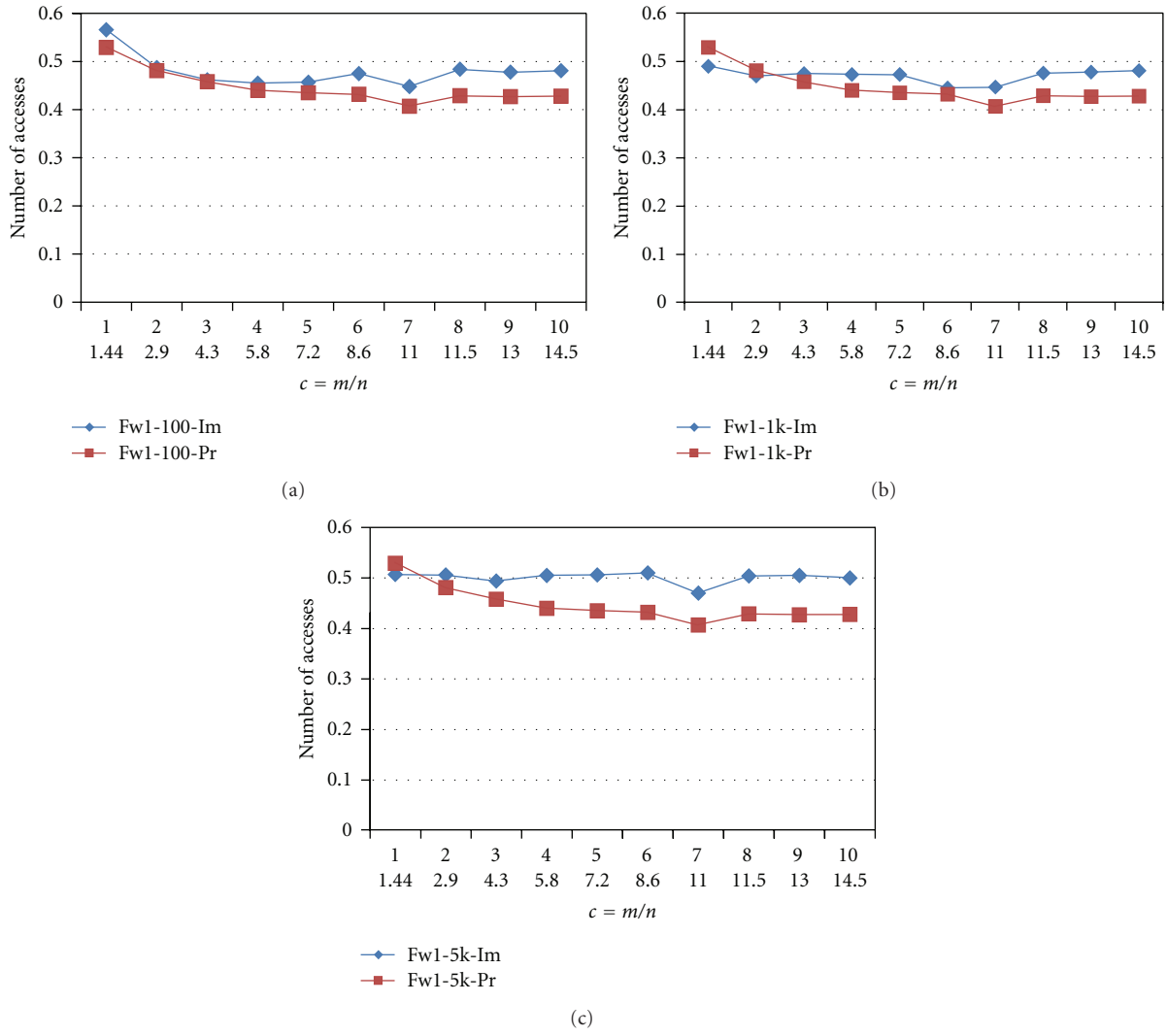

Figure 6: The number of accesses in CCBF normalized to the number of accesses in standard Bloom filter includes mathematical analysis and software implementation. (a) The number of accesses in a 3-level CCBF for Fw1-100. (b) The number of accesses in a 3-level CCBF for Fw1-1k. (c) The number of accesses in a 3-level CCBF for Fw5-1k.

In this figure, "Fw1-xx-Im" shows the graph of the software implementation, and "Fw-xx-Pr" shows the graph of mathematical analysis results that are calculated from (10). The vertical axis shows the number of accesses that are normalized to $nk(1 + p_f)$ (n is the number of items and k is number of hashing functions). The horizontal axis includes two sequences that the first one shows number of hashing functions and the second one specified by $c = m/n$ (m represents the size of address space in counter array in CCBF, and n represents the number of items) shows corresponding value of previous sequence. As an example for $k = 3$, $c = 4.3$ generates minimum false positive probability. These rule-set databases (Fw1-100, Fw1-1k, and Fw1-5k) are synthetic that were generated by the Classbench benchmark. From Figure 6, we can observe that the number of accesses is decreased for different configurations. The

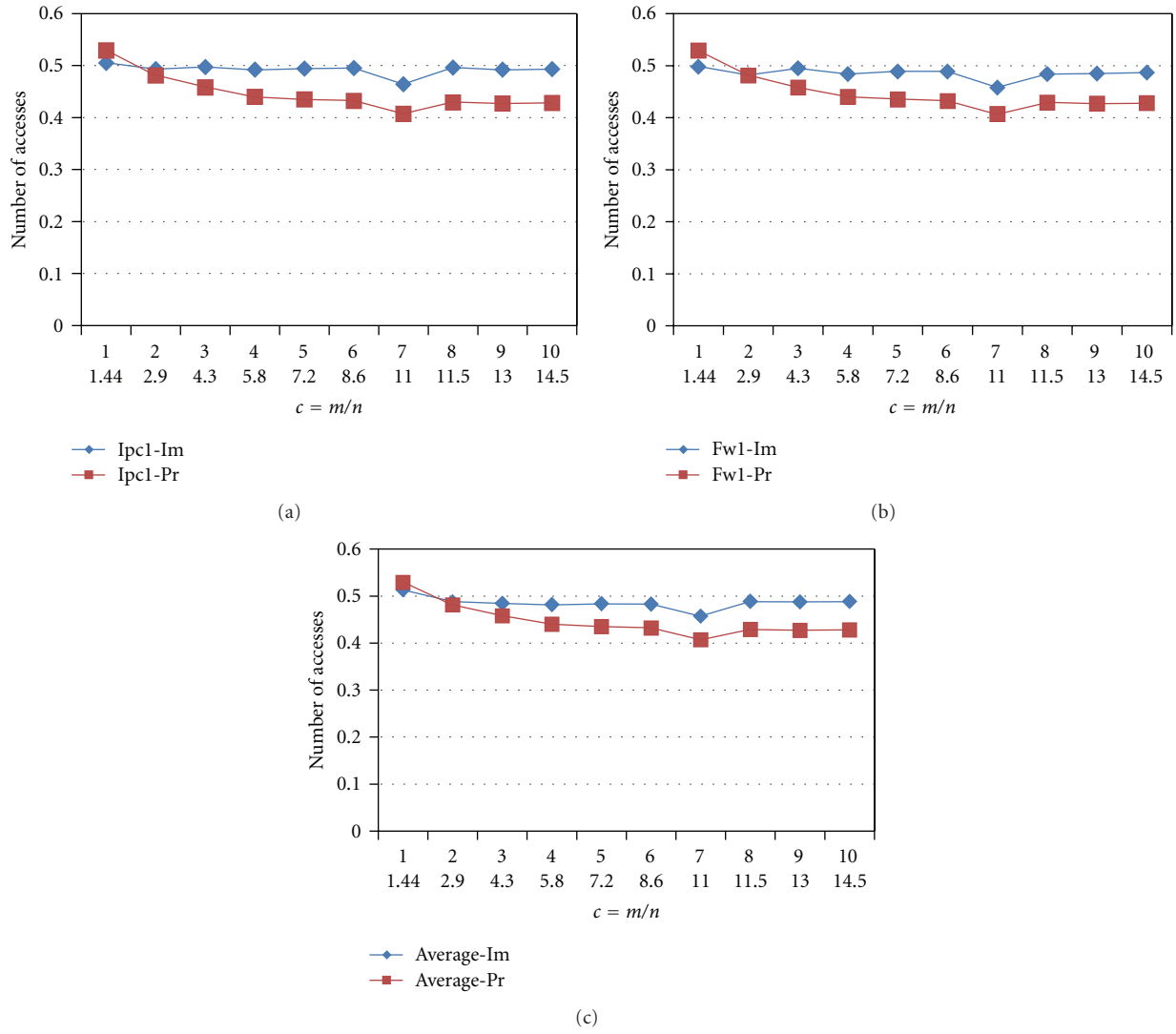

Figure 7: The number of accesses in CCBF normalized to the number of accesses in standard Bloom filter includes mathematical analysis and software implementation. (a) The number of accesses in a 3-level CCBF for Ipc1. (b) The number of accesses in a 3-level CCBF for Fw1. (c) The average number of accesses in a 3-level CCBF for all of the utilized rule-set databases in Table 1.

software implementation and mathematical analysis results for Fw1, Ipc1, and average of all rule-set databases are depicted in Figure 7.

Figures 7(a) and 7(b) depict the number of accesses for Fw1 and Ipc1 rule-set databases that were extracted from real rule-set databases. Figure 7(c) depicts the average for all utilized rule-set databases.

From Figures 6 and 7, we can observe that the mathematical analysis results are verified by the software implementation results. Based on the mathematical analysis of CCBF, the number of accesses is decreased by at least 53%. The implementation results of the software packet classifier are at most 7.8% (3.5% in average) less than corresponding mathematical analysis results. This difference is due to the following facts: number of tuples, distribution of rules inside the tuples, and utilized hashing functions. In the packet classification using tuple space, the number of tuples and the number of rules in the tuples are variable for different rule-set databases and different tuples in each rule-set database. In most of the rule-set databases, one tuple includes about half of the rules, and some tuples only have one or several rules. In the mathematical analysis, the results were obtained by investigating a CCBF with a big bit array and a single set of items. The total size of cache levels for real rule-set databases in a 3-level CCBF is depicted in Figure 8.

In this figure, "rule-set-im" shows the total size of cache for different rule-set database that the results are extracted by a software packet classifier and normalized to nk (number of items multiply by number of hashing functions). Based on the software implementation, the total cache size has some fluctuations. This is due to internal gaps of the buckets in the third level of the CCBF.

4.1. Discussion. The CCBF stores the incoming items (rules) in the memory similar to a traditional replacement algorithm

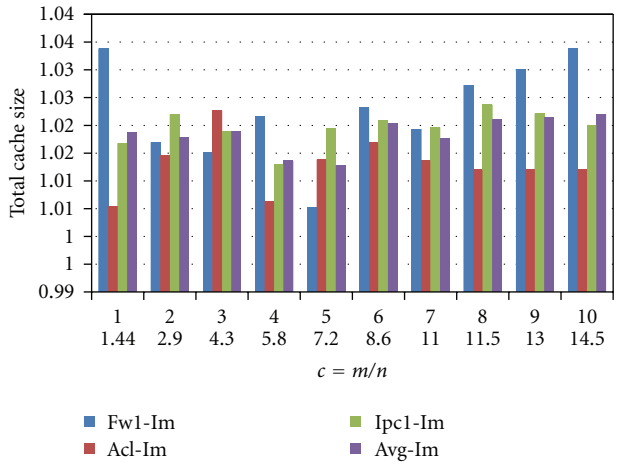

Figure 8: The total size of cache in CCBF normalized to the size of memory in the standard Bloom filter.

that is called least frequently used algorithm (LFU) [19]. In the CCBF, a bucket with larger counter has more reference; therefore, it resides in a higher cache level with lower access time, and the bucket with lower counter resides in a lower cache level. The CCBF overheads in comparison to the standard Bloom filter are managing different cache levels and the segmentation of the large buckets. In the CCBF, the bucket size of third level is set to 3 therefore, larger buckets should be segmented in to different buckets and linked together. It should be noted that the CCBF is different from the hash table with block read support. This is because, in the CCBF the block is read when the corresponding counter has value larger than one otherwise there is no need to block read. From Figure 8, we can observe some difference in the CCBF size between the software implementation and mathematical analysis results. This is because of internal gap in the buckets in the implementation of CCBF. To overcome this problem we utilize the following mechanisms:

(i) shared global overflow area

(ii) level overflow area.

A shared global overflow area is a memory space to store the overflow items. When the incoming item cannot be stored on its level, it is stored in the shared global overflow area. The second mechanism is a level overflow area that is allocated as an additional memory for each level. This solution is more practical to implement. This is because the size of each level is assumed larger than the size of the level in mathematical analysis results.

5. Overall Conclusions

In this paper, we presented a new approach to embed a multi-level cache memory in a counting Bloom filter (CCBF). Using the counting Bloom filter property, the number of accesses and sizes of the l-level and 3-level cache in the CCBF architecture were investigated. To verify the mathematical analysis results, we implemented a software packet classifier in basic tuple space using an $H3$ class of universal hashing functions. The results show that incorporating a multi-level cache memory will improve the performance of Bloom filter in comparison to a standard Bloom filter. Based on the mathematical analysis results of CCBF architecture, the number of accesses is decreased at least by 53%. The implementation results are at most 7.8% more than corresponding mathematical analysis results. We expect this approach to be useful in the design of high-performance memory architectures utilized in network processors and related applications such as packet classification and web caching.

References

[1] J. Mudigonda, H. M. Vin, and R. Yavatkar, "Overcoming the memory wall in packet processing: hammers or ladders?" in *Proceedings of the Symposium on Architectures for Networking and Communications Systems, (ANCS '05)*, pp. 1–10, 2005.

[2] S. Kumar and P. Crowley, "Segmented hash: an efficient hash table implementation for high performance networking subsystems," in *Proceedings of the Symposium on Architectures for Networking and Communications Systems, (ANCS '05)*, pp. 91–103, 2005.

[3] R. Ricci, S. Barrus, D. Gebhardt, and R. Balasubramonian, "Leveraging bloom filters for smart search within NUCA caches," in *Proceedings of the International Workshop on Complexity-Effective Design*, 2006.

[4] 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 the International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, vol. 35, no. 4, pp. 181–192, 2005.

[5] S. Dharmapurikar, H. Song, J. Turner, and J. Lockwood, "Fast packet classification using bloom filters," Tech. Rep. 27, Department of Computer Science and Engineering, Washington University, St. Louis, Mo, USA, 2006.

[6] A. Broder and M. Mitzenmacher, ""Network applications of bloom filters: a survey," in *Proceedings of the Annual Allerton Conference on Communication, Control, and Computing*, pp. 636–646, 2002.

[7] F. Chang, F. Wu-chang, and L. Kang, "Approximate caches for packet classification," in *Proceedings of the 23th IEEE International Conference on Computer Communications (INFOCOM '04)*, pp. 2196–2207, March 2004.

[8] V. Srinivasan, "A packet classification and filter management system," in *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '01)*, pp. 1464–1473, April 2001.

[9] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," in *Proceedings of the International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 135–146, 1999.

[10] H. Yu and R. Mahapatra, "A memory-efficient hashing by multi-predicate bloom filters for packet classification," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM '08)*, pp. 1795–1803, 2008.

[11] S. Dharmapurikar, H. Song, J. Turner, and J. Lockwood, "Fast packet classification using bloom filters," in *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '06)*, pp. 61–70, ACM, 2006.

[12] M. Ahmadi and S. Wong, "A cache architecture for counting bloom filters," in *Proceedings of the 15th IEEE International Conference on Networks, (ICON '07)*, pp. 218–223, November 2007.

[13] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.

[14] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network*, vol. 15, no. 2, pp. 24–32, 2001.

[15] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (ACM/SIGCOM '98)*, vol. 28, no. 4, pp. 203–214, 1998.

[16] J. L. Carter and M. N. Wegman., "Universal Classes of Hash Functions," in *Proceedings of the 9th annual ACM Symposium on Theory of Computing*, pp. 106–112, ACM Press, 1977.

[17] M. V. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," *IEEE Transactions on Computers*, vol. 46, no. 12, pp. 1378–1381, 1997.

[18] M. Ahmadi and S. Wong, "Hashing functions performance in packet classification," in *Proceedings of the International Conference on the Latest Advances in Networks (ICLAN '07)*, pp. 127–132, 2007.

[19] A. S. Tanenbaum and A. S. Woodhull, *Operating systems: Design and Implementation*, Prentice-Hall, Upper Saddle River, NJ, USA, 3rd edition, 2006.