A Throughput-Efficient Packet Classifier with n Bloom Filters

Heeyeol Yu and Rabi Mahapatra

Texas A&M University
College Station, TX 77843
Email: {hyyu,rabi}@cs.tamu.edu

***Abstract*— Packet classification is a critical data path in a high-speed router. Due to memory efficiency and fast lookup, Bloom filters (BFs) have been widely used for packet classification in a high-speed router. However, in a parallel packet classifier (PPC) of n *parallel* BFs, using all n BFs for a lookup is not throughput efficient in a high speed router. In this paper, we propose a multi-tiered packet classifier (MPC) for high throughput with the same memory size as a PPC. While a PPC of n BFs needs $\Theta(n)$ BF access complexity for a lookup, our MPC is geared to have the complexity which is probabilistically far less than $\Theta(n)$. Furthermore, by preprocessing a group of lookups in one cycle, each lookup is assigned to its associated BF at best effort, so that a higher throughput in an MPC is obtained. In simulation for flow identification with NLANR traces, we observed that, at most, 2.0 times more throughput was recorded than a PPC .**

I. Introduction

As the demand for high-speed and large-scale routers has surged, a class of fast packet processing, such as packet classification and IP lookup, have become the critical data path functions for many emerging networking applications. Those functions have enjoyed wide application in networking devices to support firewall, access control list, and quality of service in several network domains. They match a packet in high speed against a prioritized set of rules, which are made up of one or more fields for IP lookup or packet classification, respectively. For example, for Cisco GSR 12416 router of a 160Gbps rate, a packet lookup with a rule table needs to take about 2ns per packet in the worst case on the condition of a minimum 40-byte packet.

There are currently three major techniques for fast packet processing, Ternary Content Addressable Memory (TCAM), trie-based, and hash-based schemes. Among them, hash-based schemes have been favored due to power efficiency and balanced memory access againt the others. As a hash-based scheme, a Bloom filter (BF) has been widely documented in literature on networking [1–6]. A BF is essentially a generalized hash mechanism on a key set with k hash functions for *approximate* membership testing. Dharmapurikar *et al.* [3] introduced the first algorithm to employ BFs working *in parallel* for IP lookup. Similar approaches using BFs for fast lookup were applied to packet inspection application for network security [4], for packet classification identifying flows [5], and for network applications like content delivery [2].

Despite a BF's wide usage due to memory efficiency, a BF of an m-bit vector has a limitation of a false positive (f-positive). Thus, to sustain a extremely low f-positive rate and to resolve the f-positive are necessary in a high speed packet classification. In addition to the issue from a single BF, in a variety of applications that use a set of n BFs, a key (or a packet) query to n BFs has been designed in an ad-hoc manner such that they probe all n BFs. For instance, in packet classification [5] for Juniper T640 with 160 router ports, each of 160 BFs is assigned to a port for flows' record, and all BFs are to be probed to find the next output port for an incoming flow. Probing all BFs for a packet lookup in a cycle is not throughput efficient, because only one BF is actually associated with the lookup and the rest BFs can do other lookups in the cycle. Thus, to distribute lookup requests to their corresponding BFs without probing the irrelevant BFs for a lookup is throughput efficient in packet classification.

Our multi-tiered packet classifier (MPC) of n BFs provides such a lookup distribution for a higher throughput, compared to a parallel packet classifier (PPC) of n parallel BFs [3, 5–7]. A PPC accesses n BFs for one lookup every cycle while an MPC accesses n BFs for several lookups every cycle with the same BFs' memory amount as that in a PPC. To build 2-tiered BFs, for an example of an MPC, the total PPC memory is split for a pre-stage of small-sized BFs with one read port and a post-stage of large-sized BFs with k-1 read ports. Then, a small-sized BF is logically connected to two large-sized BFs, so that a forest of binary trees is built. In this forest, a lookup starts from parent BFs in the pre-stage to children BFs in the post-stage. Due to no false negative in a BF, since the lookup can proceed to two children BFs in a post-stage only if the lookup in a parent BF return positive either true or false, there is the possibility of not reaching the two large-sized BFs irrelevant to the lookup, compared to a PPC. Thus, the total number of the reached large-sized BFs for the lookup is probabilistically far smaller than n-2 depending on the small-sized BFs' f-positives in the pre-stage. Thus, the rest idle large-sized BFs in the post-stage can be utilized for other lookups. By distributing a group of lookups to corresponding BFs in a cycle at best, a higher throughput is achieved in an MPC.

Generally, a good packet classifier has to satisfy three criteria: 1) Throughput: a lookup must forward packets fast enough to keep up with the rapidly increasing line-rate. 2)

978-1-4244-2324-8/08/$25.00 © 2008 IEEE.

Memory: because memory size directly affects system cost, a lookup speed, and power dissipation, the total required memory should grow slowly with the number of rules. 3) Power: to keep the cooling system complexity reasonable, power dissipation for a lookup must scale well. Our MPCs for packet classification satisfies these criteria better than a PPC, and it is clear that an MPC is beneficial in any packet processing [3, 6, 8, 9].

This paper has the following contributions:

- An MPC, is proposed in a multi-tiered configuration of BFs with the same memory amount as a PPC.
- We show that putting multi-ports in small-sized BFs does not increase the total area cost in fabricating memory modules of small-sized BFs and large-sized BFs.
- Simulations for packet classification with NLANR traces [10] are conducted to measure throughput against a PPC.

The related works using BFs for packet processing are shown in Sec. II. In Sec. III, memory architecture for an MPC as well as the two operations, `insert` and `query`, are shown. Particularly in `query` two kinds of lookups, successful and unsuccessful, are considered and probabilities of them are calculated. In Sec. IV, with IP traces from [10], we measure power with different number of BFs. Finally, Sec. V summarizes benefits of MBFs and a future work.

II. Related Works

Packet classification is a key technology for modern high-speed routers. The packet classification goal is to identify a flow characterized with a 5-tuple of source IP (SIP), destination IP (DIP), protocol, source port (SP), destination port (DP) and to forward the flow to a corresponding output port. Several types of packet classifiers like TCAM-based and SRAM-based ones are suggested [8, 11–13]. As a hash-based approach, a packet classifier in [5] uses BFs in a parallel configuration, so that in a given packet lookup all BFs need to be checked to find the packet-associated flow in each BF and the packet is forwarded to a corresponding port where a BF returns 'yes'. However, in a high-speed lookup to a BF, the number of memory read ports in the BF is considerably large. Also, the number of BFs to be probed is as large as the number of a high-speed router's ports. Unlike the above schemes of the $\Theta(n)$ BF access complexity among n BFs, our MPC needs probabilistically less complexity than $\Theta(n)$ for a lookup

Besides BF applications for packet processing, applications of other domains have utilized the benefit of BFs, such as dynamic BF for data management [6], wide-area web caching [14], content delivery across overlay networks [2], IP traceback [15], query routing in peer-to-peer networks [9]. However, these applications simply process one lookup to n BFs in parallel resulting in the $\Theta(n)$ lookup complexity.

III. A Multi-tiered Packet Classifier with n BFs

In this section, we present the mathematics about a BF and an f-positive. We then introduce how to build an MPC and `insert` and `query` operations in a high throughput MPC.

A. Bloom Filter Theory

A legacy BF for representing a set S of n_i items (or keys) is described by an m-bit array memory with each initially set to 0. A BF uses k independent hash functions $h_0, ..., h_{k-1}$ within the range of $[0{:}m-1]$. For mathematical convenience, we make a natural assumption that these hash functions map each key in the universe to a random number uniform over the range. For insertion of each key $e_{j'} \in S$, the bits indexed by $h_{k'}(e_{j'})$ are set to 1 for $0 \le k' \le k-1$, $0 \le j' \le n_i-1$. To query that key e' is in S, k bits by k memory reads through $h_{k'}(e')$ should all be 1. If so, a BF returns 'yes' about a query of key e'. If not, then clearly e' is not a member of S. Even if a BF returns 'yes', there may be a probability of an f-positive that key y is falsely believed to belong to set S due to the random gathering of k bits of value 1 set by independent keys. As [1] claims, the f-positive probability is bounded as follows:

$$f \ge \{1 - (1 - 1/m)^{kn_i}\}^k \approx (1 - p)^k \ge (1/2)^{m \ln 2/n_i}. \quad (1)$$

The optimal k, the number of hash functions, that minimizes f is easily found $k = \ln 2(m/n_i)$. After some algebraic manipulation, it is clear that the requirement of $f \le \epsilon = 2^{-w}$, where w is called lookup precision, suggests

$$m \ge n_i \log_2(1/\epsilon)/\ln 2 \approx 1.44 n_i \log_2(1/\epsilon) = 1.44 n_i w. \quad (2)$$

From Eq. (2), the following important lemma can be derived:

Lemma 1 (Linear Property) *Linear property between m and n exists in Eq. (2) because given f requires that variable n_i is linearly proportionate to variable m.*

Also, in an optimal configuration, k becomes w according to Eq. (2) and the derivation that $k = m \ln 2/n_i = w$, and to be a scheme of an f-positive-free $O(1)$ lookup processing 500M packets a second for a 160Gbps router, k needs to be at least 29 ($\approx \log_2 1/500M$). Each hash function corresponds to one random lookup in an m-bit memory, and a BF with k hash functions needs the exact same k of memory read ports in an m-bit memory. Thus, a BF is considered as a high computation element due to the large value of k for the high-speed router, so that using few number of BFs for a lookup is preferable.

B. Building a Multi-tiered Packet Classifier

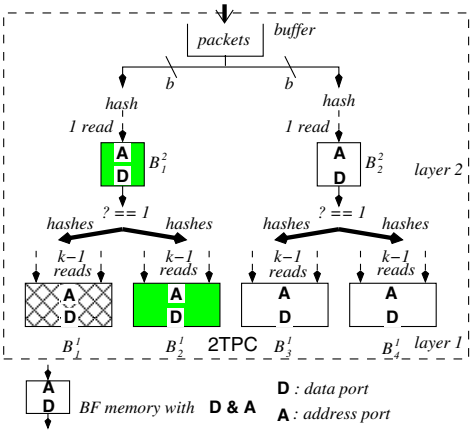


Fig. 1. Memory architecture of a 2TPC in a forest and *in pipeline*. B_j^i means the j-th BF at layer i. n=4. k=w due to an optimal k. Buffer size b=1.

In this section, we derive mathematical proof that an MPC uses the same memory size as a PPC while the detailed insertion and query are mentioned in Secs. III-C and III-D. Fig. 1 shows a configuration example of an MPC, a 2-tiered PC (2TPC) on top of 4 BFs, in place of a PPC configured in a linear form. Given desired f-positive $f=2^{-w}$, the total PPC memory in bits with n BFs is $n{\cdot}m$, where m is a BF's memory based on Eq. (2). However, with *linear property* between m and n_i and an additive operation on memory size m_t, we can reconfigure BFs in a $(r+1)$-tiered way, $r>0$, while the same memory size, m_M, for an MPC is used as follows:

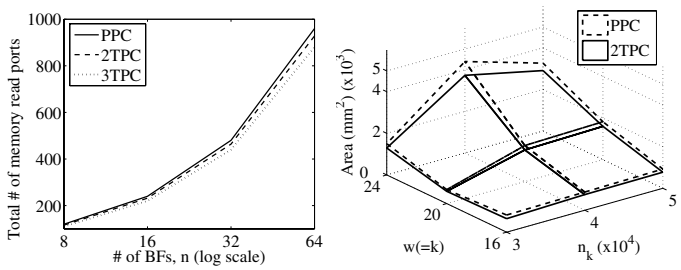
$$\begin{aligned} n \times m &= n \times \{1.44 \cdot n_i \cdot \log_2(1/f)\} \\ &= n \times \{1.44 \cdot n_i \cdot w\} = n \times \{1.44 \cdot n_i \cdot (w - r + r)\} \\ &= n \cdot 1.44 \cdot n_i \cdot (w - r) + \textstyle\sum_{t=1}^{r} \{n \cdot 1.44 \cdot n_i \cdot 1\} \\ &= \textstyle\sum_{i=1}^{n} (1.44 \cdot n_i \cdot (w\text{-}r)) + \sum_{t=1}^{r} \sum_{i=1}^{n/2^t} (1.44 \cdot (2^t n_i) \cdot 1) \qquad (3a) \\ &= m_1 + \textstyle\sum_{t=1}^{r} m_{t+1} = m_M, \end{aligned}$$

where m_t is the total memory of BFs on layer t, $r+1$ is the number of tiers, $2^t n_i$ is the number of keys in B_i^t, and the lookup precisions of a BF on layer 1 and t, w_1 and w_t, are w-r and 1, respectively. Based on Eq. (2), the f-positives of BFs on layer 1 and 2 in a 3TPC are expected to be $2^{-(w-2)}$ and 2^{-1}, respectively, and the second term in Eq. (3a) is the sum of small-sized BFs from layer 2 to layer $r+1$. Also, a BF from layer 1 covers n_i elements, and a BF from layer 2 covers $2n_i$ keys. Generally, B_i^j covers all keys from B_{2i}^{j-1} and B_{2i+1}^{j-1}, $1 \le i \le n/2$, $1 < j \le r$ in an MPC.

Although the above derivation shows that an MPC has the same memory size as a PPC, processing a lookup in small-sized BFs of one read port does not provide a higher throughput in large-sized BFs on a lower layer. For instance, even if b in Fig. 1 is set to 2, a one-read-port BF on layer 2 cannot process 2 lookups in one cycle. Thus, the number of read ports in the small-sized BF needs to be the same as b. As suggested in [3], using mini-BFs with few read ports is the solution without degrading lookup accuracy. However, even if a BF is broken into several mini-BFs, the total number of read ports in the mini-BFs is the same as that in a PPC. Thus, breaking a BF into mini-BFs only gives the possibility of fabricating BFs for packet processing, not the benefit of high throughput. However, our MPC has two benefits of few number of read ports and an area cost which can lead to fabricate small-sized BFs of multi read ports for a high throughput without area overhead.

Fig. 2 shows such two benefits, the smaller number of fabricated read ports and the smaller area for a 2TPC. Fig. 2(a) shows the required numbers of read ports in fabricating a different number of BFs for a PPC, a 2TPC, and a 3TPC, respectively. In fabricating, a 2TPC and a 3TPC use 4 and 10% few number of read ports than a PPC in all cases. Fig. 2(b) shows 2TPC and PPC area costs in a different number of w and n_i, and we measured the area costs using 4 mini-BFs for a BF in each case through CACTI [16].

Now, we show how to fabricate multi-ports in a small-sized BF without hardware overhead. There is a noticeable gap between meshes in Fig. 2(b), and the reason is that fabricating multi-ports in a small-sized memory does not need area as much as in a large-sized memory. Due to page limit, we did not plot the area costs for 2 through 5 read ports in a small-sized BF memory on layer 2. However, there is a small area increase for the multi-port memory, compared to the area of a PPC. Thus, it is clear that the buffer size b can amount to 5 at most. Also, utilizing dual reads on falling and rising edges in a clock [17] can double memory read capacity for a lookup throughput as a double data rate scheme does in DRAM and AMD Athlon64. Thus, the buffer size becomes twice and the maximum b is 10 without memory overhead in a MPC.



(a) The read port number (b) The area cost (w_2=1)

Fig. 2. (a) The total number of read ports in different number of BFs. w_3=w_2=1, w_1=13 for a 3TPC. w_2=1, w_1=14 for a 2TPC. f=2^{-15}. (b) 2TPC and PPC area costs with n=8 in .13μm process technology.

C. `Insert` in an MPC

Procedure `insert`

Input: Key e and index i for a BF on layer 1
Output: Encoded 2TPC about e

```
1 for layer j = 1 to s do
2     for t = 0 to k_j − 1 do        // h_t is t-th hash func.
3         B^j_⌊i/2^(j-1)⌋[h_t(e)]=1;    // B is Mem. on SRAM for BF
4     end
5 end
```

`Insert` operation of a key in a BF on layer 1 is as simple as insertion of the key in a legacy BF as shown in Sec. III-A. Similarly, on layer j, if a key to hash is assigned to B_i^j, the key is given to $B_{\lfloor i/2 \rfloor}^{j+1}$ for `insert` operation, $1 < j \le s$. The detailed procedure is shown in **Procedure** `insert` which does k_j times memory write on layer j. Therefore, the memory write complexity of one key insertion is $\sum_{t=1}^{s} k_t = w = k_P$ which is the same as a PPC, where k_P is based on an optimal k. Also, note that the first vertically lined **for** can be in pipeline because BF memories on a layer are independent ones from other layers. Thus, in every cycle one key insertion is performed on the condition that B_i^1 on layer 1, $1 \le i \le n$, supports multiports.

D. `Query` operation in an MPC

Unlike `insert` operation where only involved BFs need to be accessed, `query` operation needs to access all BFs to find which BFs return 'yes'. Because except one BF the rest BFs give f-positives leading to packet misclassification, the irrelevant BFs in an MPC are geared not to be probed, so that the BF access complexity in processing a lookup with n BFs

is far less than n. To provide such a complexity, we split the memory of a PPC into small-sized BFs and large-sized BFs in multi-tiers, and they are connected in binary trees. Then, accesses to large-sized BFs are made only if their parents of small-sized BFs return 'yes' (or value 1 in D) as in Fig. 1. Also, BFs in multi-tiers can be in pipeline so that there is no performance degradation. Before the detail procedure, let us introduce true- and false-path definitions in an MPC.

In `query` operation among a forest shown in Fig. 1, a true path, t-path, occurs. It is composed of shadowed BFs from a root of a tree to return 'yes'. These were involved in the previous `insert` operation for a key. For example, if a key is assigned to set B_2 in a PPC, the BFs on a t-path for a 2TPC are B_1^2, B_2^1 as shown in Fig. 1. From the above definition, in `query` operation all BFs on the t-path should return 'yes' for a given key as a legacy BF returns 'yes' because each BF has the key as a member.

Unlike a t-path, a false path is made from a group of BFs giving f-positives so that packet misclassification occurs. The f-positives by the BFs, neither stemming from a branch of a t-path nor being a complete path in a tree among a forest, can not contribute a false path, or f-path, by the definition. Also, the number of f-paths means the number of packet misclassifications. An important fact is that the probability of misclassification for an f-path contributing one packet misclassification is cumulatively calculated in product of each f-positive on the f-path.

1) False classification in a successful lookup: We divide a lookup in two ways: 1) a successful lookup and 2) an unsuccessful lookup. If a queried key in a lookup exists a BF member, we call the lookup an SL. Now, we show the misclassification probability in an SL.

By a recursive definition, the probability $P_a(i)$ that root a in a binary tree has i packet misclassifications is the product of the following three: the probability of an f-positive in root a of the binary tree, the probability that a left subtree has i-j packet misclassifications, and the probability that a right subtree has j packet misclassifications as the following:

$$P_a(i) = \sum_{j=0}^{i} f_a \times P_l(i-j) \times P_r(j), \quad (4)$$

where f_a is the probability of an f-positive from BF a, and as a base case, $P_{B_1^1}(1)=f_{B_1^1}$. Finally, the dominant probability, $\mathcal{P}_s(1)$ that a single packet misclassification occurs across a forest is the following:

$$\mathcal{P}_s(1) = \sum_{j=1}^{r-1} P_{B_t^j}(1) + \sum_{i=2}^{n/2^{r-1}} P_{B_i^r}(1), \quad (5)$$

where r is the number of tiers, the first term is the summation of Eq. (5)'s probabilities about BFs attached on a t-path and the second term is the summation of probabilities about the remaining trees among the forest.

2) False classification in an unsuccessful lookup: Since all packets are not under specific flows based on a flow table, a UL is important as much as an SL. Unlike an SL, in a UL there is no t-path. This means that what a BF returns, if any, is an f-positive. The dominant probability, $\mathcal{P}_u(1)$ that a single packet misclassification happens in a UL is

$$\mathcal{P}_u(1) = \sum_{i=1}^{n/2^{r-1}} P_{B_i^r}(1). \quad (6)$$

Procedure `query`

Input: Forest F of binary trees for an MPC and key e
Output: Set S for a true path and a group of false paths

```
for tree T ∈ in forest F do
    S = S ∪ query_BT(T, e) ;
end
return S;
```

Procedure `query` shows the details of query operation on an MPC. The code in the vertical line of **Procedure** `query` can be implemented *in parallel*. Also, it calls subroutine `query_BT` which is working *recursively* and *in pipeline* on each layer in a binary tree to check a BF for the key e as a legacy BF does. Also, pipelining on layers in a binary tree makes it sure that the `query` complexity is $\Theta(1)$ as a PPC' complexity is.

Based on Eqs. (5) and (6), the expected packet misclassification considering SL and UL rates is

$$p_s \sum_{i=1}^{n-1} i \cdot \mathcal{P}_s(i) + (1-p_s) \sum_{i=1}^{n} i \cdot \mathcal{P}_u(i) = p_s \cdot E_s + (1-p_s)E_u, \quad (7)$$

where p_s is an SL rate, and E_s and E_u are the average packet misclassifications for an SL and a UL, respectively.

There is a minuscule classification performance degradation in using an MPC. Fig. 3 shows the average packet misclassification of a PPC and a 2TPC based on Eq. (7) with a rate of successful lookup p_s. There are three important facts to consider: 1) Given desired f-positive, f, as long as the n is larger, the value of the average packet misclassification is getting larger due to bigger binomial coefficient value $B(f,n)$. 2) Given the same memory size, the probabilities of PPC-n and 2TPC-n for a UL are the same while in a dominant rate of an SL, there is a minuscule difference, 2E-9, between them. 3) The difference gets smaller as long as the n is larger. In conclusion, as long as the number of BFs, n, and the rate p_s are larger, the difference of packet misclassifications between a PPC and a 2TPC is negligible. The one-packet misclassifications of Eqs. (5) and (6) show the same phenomenon shown in Fig. 3.

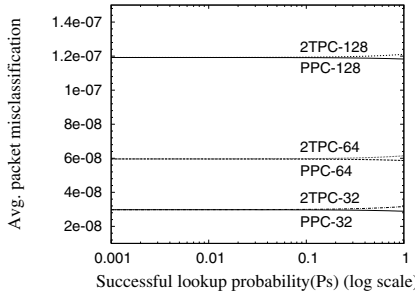


Fig. 3. The average packet misclassification for a PPC-n and a 3TPC-n in a different SL rate. f=2^{-w}=2^{-30}, w_1=28, w_2=w_3=1. $n \in \{32, 64, 128\}$.

`Delete` operation is not as easy as `insert` because a basic BF in [3, 5–7] does not support deletion of a key which was encoded in the BF. If a low power counting BF (L-CBF) [18] is adopted, the operation of `delete` is as easy as a legacy BF.

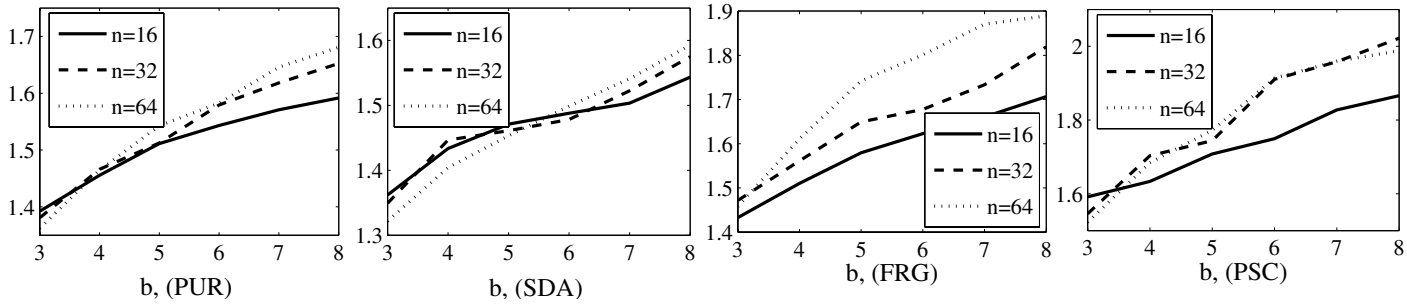


Fig. 4. Throughput ratios of a 2TPC against a PPC in four traces with different number of buffer size b and n BFs. w_1=28, w_2=2.

IV. Simulation Environment and Result

We measured PPC and MPC throughputs with IP trace from NLANR PMA [10]. We assume that a PPC needs one cycle to process a packet lookup to n parallel BFs while an MPC can process a group of packet lookups in one cycle. The throughput is defined as the number of packets over the number of simulation cycles to process the whole IP traces. The IP traces we used are PUR, SDA, FRG, and PSC which have 19.4K, 29.5K, 39.7K, and 37.9K flows as rules, respectively. In simulation, we tested 193.3K, 292.2K, 337K, and 314.3K packets in flow identification with different number of router ports, each having the same number of flows equally.

Fig. 4 shows the average throughput ratios of four traces by 10 runs in a 2PC architecture where each small-sized BF on layer 2 has a b-sized buffer to process b packets in the buffer in one cycle. Once they process packets in the their buffers, the results are forwarded to large-sized BFs on layer 1. A BF on layer 1 works on a partially processed packet only if a parent BF of the BF returns 'yes' to the packet. Thus, if a BF on layer 2 returns 'no' about a packet, the children BFs of large size can process other following packets, leading to a higher throughput. In each subfigure, in all different numbers of BFs the larger is the buffer size, the higher throughput ratio is, proving that our MPC gives a higher throughput performance than a PPC. At most 2.0 times throughput was observed in PSC. Even if we simulated a case of, at most, 64 BFs, if a larger number of BFs and buffer size b is used, our MPC is believed to show higher throughput than those in Fig. 4.

V. Conclusion and Future Work

For high-throughput packet processing, we have suggested an MPC which reconfigure BFs into small-sized BFs and large-sized BFs in a multi-tiered way without memory overhead, compared to a PPC. By Lemma 1 in Sec. III-A, we showed how to build an MPC with the same memory amount as a PPC in Sec. III-B. It is observed that the number of fabricated read ports in BFs' memory as well as the area cost for an MPC are reduced with the same memory. Also, we showed the `insert` and `query` as simple as those of PBFs. Finally, with the same memory amount as a PPC, our MPC provides the same probability of an f-positive for a UL and a minuscule increase in the probability for an SL as shown in Fig. 3. In simulation with NLANR's IP traces for flow identification, an MPC shows a higher throughput efficiency for all traces than a PPC, at most 2.0 times. In future work, we will consider power efficiency in an MPC.

References

[1] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, no. 4, 2002.
[2] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed Content Delivery Across Adaptive Overlay Networks," in *SIGCOMM '02*.
[3] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor, "Longest Prefix Matching using Bloom Filters," in *SIGCOMM '03*.
[4] S. Dharmapurikar and J. Lockwood, "Fast and Scalable Pattern Matching for Network Intrusion Detection Systems," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 10, pp. 1781–1792, 2006.
[5] W.-c. F. F. Chang and K. Li, "Approximate Caches for Packet Classification," in *INFOCOM 2004*, pp. 2196–2207.
[6] H. C. Deke Guo, Jie Wu and X. Luo, "Theory and Network Applications of Dynamic Bloom Filters," in *INFOCOM 2006*.
[7] T. S. Sarang Dharmapurikar, Praveen Krishnamurthy and J. Lockwood, "Deep Packet Inspection using Parallel Bloom Filters," in *MICRO 37: Proceedings of the 37th annual ACM/IEEE international symposium on Microarchitecture*, 2004, pp. 52–61.
[8] H. Song, J. Turner, and S. Dharmapurikar, "Packet Classification Using Coarse-grained Tuple Spaces," in *ANCS '06*, pp. 41–50.
[9] J. X. A. Kumar and E. Zegura, "Efficient and Scalable Query Routing for Unstructured Peer-to-Peer Networks," in *INFOCOM '05*, pp. 13–17.
[10] Paasive Measurement and Analysis Project, National laboratory for Applied network Research (NLANR). [Online]. Available: http://pma.nlanr.net/Traces/Traces
[11] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and Scalable Layer Four Switching," in *SIGCOMM '98*.
[12] K. Lakshminarayanan, A. Ranga, and S. Venkatachary, "Algorithms for Advanced Packet Classification with Ternary CAM," in *SIGCOMM '05*.
[13] M. S. Jun Xu and J. Degroat, "A Novel Cache Architecture to Support Layer-Four Packet Classification at Memory Access Speeds," in *INFOCOM 2000*, pp. 1445–1454.
[14] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary Cache: a Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, 2000.
[15] A. C. Snoeren, "Hash-based IP Traceback," in *SIGCOMM '01*, pp. 3–14.
[16] S. Wilton and N. Jouppi, "An Enhanced Access and Cycle Time Model for on-chip Caches," DEC Western Research Lab, Tech. Rep., 1994.
[17] D. A. Patterson and J. L. Hennessy, *Computer Architecture: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990.
[18] A. M. Elham Safi and A. Veneris, "L-CBF: a Low-Power, Fast Counting Bloom Filter Architecture," in *ISLPED '06*, pp. 250–255.