Bulletin of Electrical Engineering and Informatics
Vol. 12, No. 2, April 2023, pp. 958~968
ISSN: 2302-9285, DOI: 10.11591/eei.v12i2.4202

Cuckoo filter based IP packet filtering using M-tree

Aladdin Abdulhassan[1], Roaa Shubbar[2], Mohammad Alhisnawi[1]
[1]Department of Information Network, Faculty of Information Technology, University of Babylon, Babylon, Iraq
[2]Department of Software, Faculty of Information Technology, University of Babylon, Babylon, Iraq

Article Info

Article history:

Received Jun 4, 2022
Revised Aug 18, 2022
Accepted Aug 28, 2022

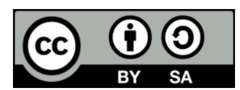
Keywords:

Approximate membership query
Cuckoo filter
IP packet filtering
M-tree
OpenFlow
Software defined networking

ABSTRACT

Internet protocol (IP) packet filtering as a firewall (FW) technology is one of the most widely researched networks functions over the past two decades. IP packet filtering is the process of filtering incoming and outgoing network packets by matching several packet headers fields with thousands of predefined filters known as filter-set. With the development of modern network technologies such as software-defined networking (SDN) and the increase in attacks threatening network security, attention has become focused on IP packet filtering. With the growing size and number of filter-sets, it becomes a challenge to perform IP packet filtering at wire-speed. In this paper, a new method is proposed for IP packet filtering, where two data structures were combined to produce a new data structure suitable for IP packet filtering with high performance and support dynamic access to filters as well as support approximate membership query. Experimental results show that the proposed method has a high throughput of 10.8 mega packets per second (MPpS) with high filtering accuracy and low memory requirements to working on big filter-sets (up to 1 mega filters).

This is an open access article under the CC BY-SA license.



Corresponding Author:

Aladdin Abdulhassan
Department of Information Network, Faculty of Information Technology, University of Babylon
Babylon, Iraq
Email: aladdin.alsharifi@uobabylon.edu.iq

1. INTRODUCTION

In recent years, a modern networking approach called software-defined networking (SDN) has emerged to provide modern network services that traditional networks cannot provide such as security, policy routing, and quality of service (QoS). The increasing demand for such services and the significant expansion of network size make traditional network technologies insufficient for network management. The availability of such services in the network is accompanied by high security threats. Internet protocol (IP) packet filtering represents one of the most important security function in modern networking technologies [1]–[3]. In general, IP packet filtering is the process of controlling access to a network, analyzes incoming and outgoing network IP-packets to make a decision to allow or block their passage according to their source and destination IP addresses (SIP and DIP) [4]. It works by setting up a filter set in a network firewall (FW) device to examine each incoming/outgoing packet and inspect their SIP and DIP addresses and some other packet header fields.

The operation of the IP packet filter in SDN is slightly similar to the work of a traditional network FW. The main difference is that in SDN, the control plane (routing process) is separated from data plane (forwarding process), so that the control work (including filtering) is performed by a single virtually centralized controller. The controller installs a set of filters called filter-set or rule-set in a flowtable using OpenFlow protocol. Each filer consists of a set of fields in addition to action and priority [5]. The same action is set to all incoming packets whose header fields are match to a corresponding filter, in case of more than one filter are matched to one packet, the filter with highest priority is selected [6]. Instead of classical

prefix matching in the traditional networks, in SDN, packet header fields are matched with up to 40 fields in OpenFlow protocol version 1.5.1 [7]. This matching is a very long process as each packet is matched with thousands of filters in the filter-set, as well as each filter contains fields of different types where some of them require exact matching and some of the other require range matching in addition to prefixes matching. With the increasing demand for high internet performance and QoS, getting all these tasks done at current wire speed is a challenge.

In this paper, a space-efficient data structure (CM-tree) has been proposed to build our IP packet filtering approach. In our CM-tree, a probabilistic data structure (cuckoo filters (CF) [8]) have been integrated with a dynamic access tree data structure (M-tree [9]). According to Ciaccia *et al.* [9] M-tree is an indexing structure was proposed. It was built to index similarities in general metric areas as its composition depends on metric areas function. M-tree is a hierarchical multi-branch tree that has the ability to add new data or delete it dynamically. M-tree nodes are hyper-sphere-shaped nodes. Each node has up to a T pre-defined threshold limit of entries. Each node entry define the routing element as the center of its hyper-sphere, and the allowed distance as its hypersphere radius r. All the internal subnodes data belong to a specific node must have the form of (d, r), where the distance d to that nodes most be smaller than or equal to its parent radius r Figure 1, while leaf nodes contain links to real data. M-trees are balanced semi-fulled tree so that all of the leaf nodes are at the same level and the tree grows bottom-up.

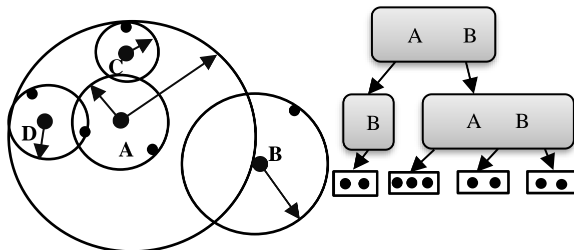


Figure 1. Structure of an M-tree

To insert a new object to the M-tree, four main steps are needed:
- Finding a convenient place in a leaf node to insert the new object.
- If the selected leaf node is already full (has K entries), split that node into two new nodes and distribute its content among these new nodes and link them to its parent node.
- If the parent node is already full (has k entries), perform step 2 on the parent node and so on till the root.
- If the root node needs to be split, create new root as a parent to the two new nods from the old root and increase the M-tree height by one.
- Finding a convenient place in M-tree to the new nodes is performed as following:
- At each level, we chose a node entry (Object) (O_j) that a covering area can cover the rout object O. That means the distance d between the rout object (O_j) and the query object O is less that the rout object's radius $r(O_j)$.

$$d(O_j, O) < r(O_j) \quad (1)$$

- If there are more than one rout object can satisfy (1), chose closest one to the rout object O.
- If there is no rout object can satisfy (1), expand the covering radius of one of the closet rout objects, then, chose the rout object with minimum expansion of rout objects radius.

The M-tree best query time is O(log n) and can store up to M filters in each of its nodes, and CF query time is O (1). The CF [8] can be considered as a compact, approximate set-membership data structure that have the ability to perform elements insertion/deletion in O (1) time. Essentially, it depends on its work on utilizing extremely compact cuckoo hash table [10] that stores fingerprint rather than the original element. Cuckoo hash table holds an array of buckets where every element (x) has two candidate buckets indexed by the values of two hash functions hash 1(x) and hash 2(x). The main disadvantage of using trees data structures in such system is their high memory requirements and its the main purpose to integrate CFs in M-tree nodes in our proposed structure. CFs have been used to overcome the high memory requirements, the filters from the filter-set have been hashed to CFs and then, these CFs are integrated to their corresponding nodes in the M-tree.

The source and destination IP addresses from the filter-set have been used to construct the M-tree hyper-spheres, and the rest of the fields were grouped and hashed into CF, and then that CF was integrated into the relevant M-tree node. Resulting CM-tree has a high query performance that it gained from its component M-tree, as the M-tree is balanced, full, or nearly full and also has the efficiency of using storage due to integrated CFs that store fingerprints of fields instead of storing the fields themselves. In addition to storage reduction purposes, CFs have been used to reduce query time as the CFs block a lot of unnecessary queries in the inner branches of the M-tree in particular those that match the M-tree hyper-spheres of specific nodes but do not get matched in the integrated CFs of those nodes.

A lot of software and hardware solutions have been introduced to solve the traditional IP packet filtering problem. Hardware methods such as ternary content addressable memory (TCAM) [11] and bitmap intersection [12] are fast and efficient because they use special hardware. Usually, such types of hardware are expensive, and, so, the hardware methods lack scalability and portability. Many researchers went to programmatic methods to solve the problem of IP packet filtering, which has the advantage of scalability and portability [13]–[16].

Modern network technologies such as SDN and network virtualization had shifted the classical five-field IP packet filtering to many-field IP packet filtering. It also shifted researchers focus in their recent research to the problem of many-field packet filtering. Therefore, a group of close research studies concerned with the problem of many-field packet filtering and classification were studied for the purpose of analysis and comparison.

To improve the search time, Lu and Sahni [17] presented two versions of the binary search on levels binary search on levels (BSOL) algorithm (BSOL1, BSOL2), in these algorithms a decision trie was introduced to implement the binary search on prefix lengths algorithm [18]. They stored the set of filters in one or two treis according to the version of algorithm in a static manner. Their trie can be used with any field type in the filter sets including (prefix and range). In their work, each filter has represented as d-dimensional hyper-rectangle depending on number of fields in each filter in the filter-set. Then, they build their decision tree from these d-dimensional hyper-rectangle where the trie root corresponding to the whole space and the same space is divided equally between its two children. Cheng and Wang [19] studied the BSOL algorithm and introduced enhanced version called BSOL with replication control (BSOL-RC). In their algorithm, they tried to minimize the number of replicated filters by performing a replication rule with the BSOL algorithm. Lim and Byun [20] build a packet classifier by integrating a bloom filter into an area based quad tree (BF-AQT). The internal nodes with rules have been pushed into leaves to get better performance. A bloom filter has been set to each filled node in their tree and linked to a linked list in the filter-set, while the other nodes have hashed to a hash table. A scalable parallel approach has been proposed in [21], in this approach, the authors have used range tree [22] along with hash tables [23] to organize the filter-set. Every partial result has been recorded with filter ID, and merge algorithm has been used to combine those partial results and draw the final result.

In the proposed work, the results of the experiment showed that the proposed method has high filtering throughput up to 10.8 mega packets per second (MPpS), needs a relatively small storage memory, and has a high filtering accuracy. The main contribution of this paper was as follows: first, the proposal of the CM-tree as a hyper-sphere-shaped nodes tree in which CFs have been integrated into its nodes to help reduce the amount of storage required and speed up query operations. Then the modification of the CF to a dynamic length filter to facilitate its use without sacrificing redundant storage due to the large differences between the number of integrated elements in the inner and outer tree nodes. And finally, the resulted CM-tree was used for the first time in IP packet filtering, and the results were very encouraging, as a comparison was made with three related works using several evaluation metrics and the results showed that the proposed method provides better performance in almost all cases in terms of all the evaluation metric.

2. THE PROPOSED SYSTEM ARCHITECTURE

In the proposed method, CF has been integrated with M-tree to generate a new structure to organize the filter-set in new tree format. The proposed CM-tree has the ability to minimize the required memory access during the query time by pruning most of the redundant tree nodes. SIP and DIP addresses pair from each filter in the filter-set have been formed as a tree-node entry to build the M-tree, while the remaining fields, have been hashed to a CF then inserted to the corresponding tree-node. Figure 2 shows the main structure of the proposed filterer, we have described each of this block diagram in the following subsections.

2.1. Hyper-spheres creation

The first step to build the proposed CM-tree structure is to convert the filters from filter-set into hyperspheres to represent the tree nodes entries. As can be seen in Figure 2 each filter consisting of many

fields. For each hyper-sphere, there are some information are needed including its identifier rendezvous identifier (RID), center CENTER, radius R, distance to its parent node distributed topology processing (DTP), and link to the actual filter *Ptr. To convert the filters into hyper-spheres, we need to exclude the SIP & DIP addresses from each filter, then we need to convert these prefixes addresses into ranges values. The lower and upper limits of the prefixes have been used as the range limit. After that, the CENTER will be (SIP range/2, DIP range/2). The radius can be calculated by using Pythagoras' theorem as showed in (2), and the distance to its parent node DTP will be assign at the insertion time, and a link to the corresponding filter is generated. To understand the hyper-spheres Creation process more clearly, a simple filter-set of 7 filters have been considered as shown in Table 1. Only the required fields of this process have been showed. A unique character has been assigned for each filter as RID, the SIP and DIP for each filter have been converted into range values, and the hyper-spheres center and radius have been calculated (2).

$$R = \frac{\sqrt{(SIP\ range\ size)^2 + (DIP\ range\ size)^2}}{2} \quad (2)$$

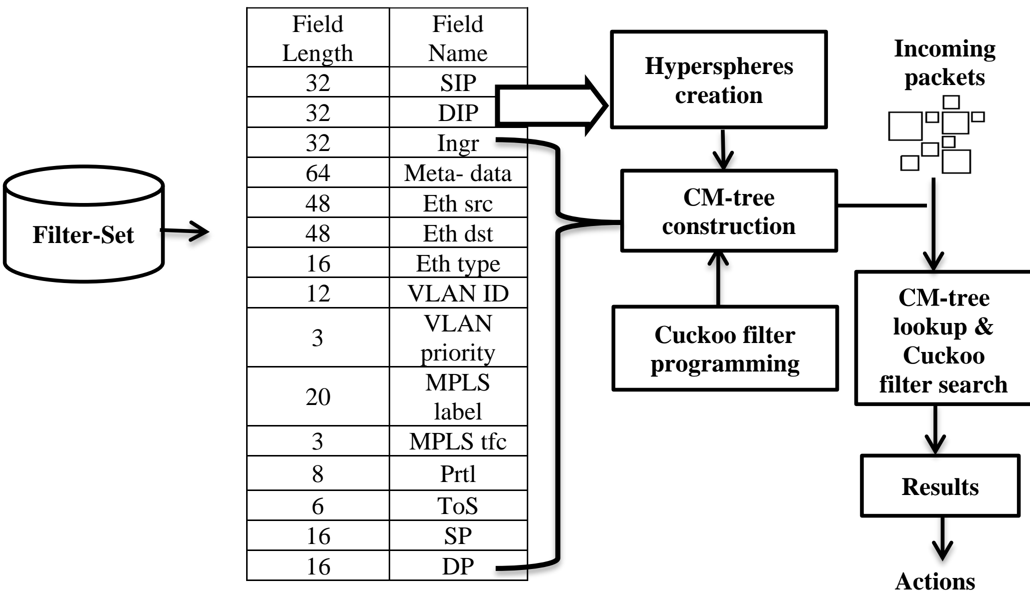


Figure 2. Block diagram of the proposed system

Table 1. A simple example of 12 filters with SIP and DIP, and prefix to range conversion

RID	SIP	R. Start	R. End	DIP	R. Start	R. End	Center	Radius
A	-	-	-	-	-	-	96,96	80
B	-	-	-	-	-	-	216,223	24
C	-	-	-	-	-	-	204,168	91
A	01*	64	127	01*	64	127	96,96	45
D	00011*	24	31	011*	96	127	28,112	16
E	01*	64	127	00111*	48	63	96, 54	32
C	11001*	200	207	1010*	160	175	204,168	8
F	01100*	96	103	1101*	208	223	200,2016	8
B	01*	64	127	11011*	216	223	96,220	16
G	111*	224	255	11011*	216	223	240,220	16
H	11*	192	255	011*	96	127	224,112	35

2.2. M-tree construction

The hyper-spheres have been inserted into the M-tree one by one. The M-tree optimization of [24] has been used so to insert an entry e into a node n, the parent-child distances between n and e has been computed and stored along with the child-node pointers. This value is very useful in the tree query process and can be used in conjunction with the triangle inequality to bound the range of the distance of child node to query object without real computation. To insert a new element e to an empty M-tree, we need to find the

best suitable leaf node to accommodate entry e, in case of node overflow, we need to perform node splitting procedure.

To find the best leaf node, we need to find the best suitable rout-node entry (Ej) at each level to follow a path in the M-tree which would avoid any enlargement of the covering radius, (i.e. to insert a new entry e into M-tree), at each level, find the rout-entry(Ej) that satisfy the triangle inequality d(Ej+r(E))≤r(Ej) so that the distance d(Ej, E) plus the new entry radius r(E) should be smaller or equal to the rout entry radius r(Ej). If there are more than one suitable rout-node entry Ej, we chose the best one that is closest to the routing entry Ej. In case of no routing-node entry has the property of (2), then we need to perform covering radius magnification. In such case, we have to chose a routing node that can minimize the increasing of the covering radius. The insertion process for the tree was represented from the example in Table 1 a graphical representation in Figure 3, in this example an M-tree with degree M=3 was created so that 2≤m≤3 entries can be stored in each routing-node of this tree as shown in Figure 4 step by step.

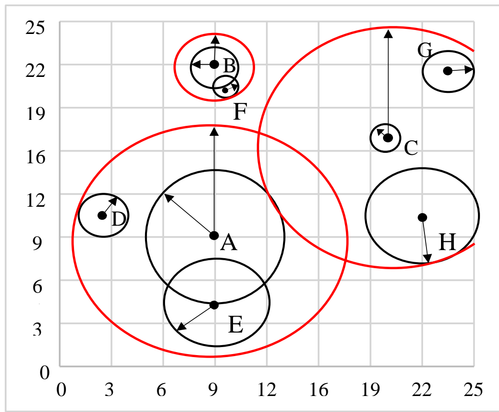


Figure 3. Graphical representation of the filters from Table 1

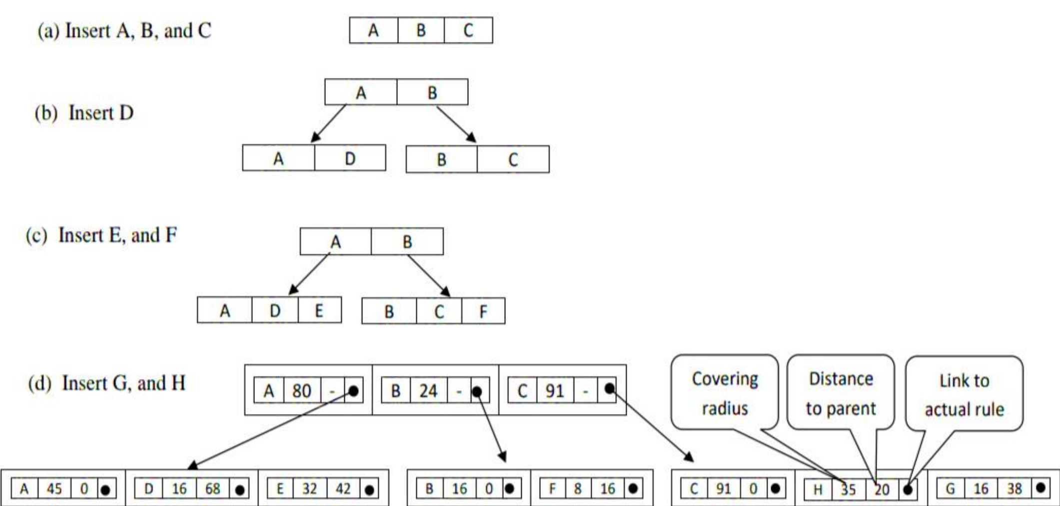


Figure 4. Graphical representation of insertion the hyper-spheres from Table 1 into an empty M-tree

In case of the chosen routing-node is already full, a node split is necessary (i.e. partition the routingnode entries plus the new object entry into two new nodes N1 and N2). Two entries from the old entries that have maximum distance between them among all other pairs are selected as seeds and assigned to N1 and N2. Then the remaining entries are assigned to N1, N2 according to which of N1, N2 seeds has minimum distance to these entries. The new two nodes are assigned to the old node parent. The split procedure is propagated to upper levels until the root whenever is needed, and if the splitting is necessary to

the root level, the tree high is increased one time. The distance of all entries in the new formed nodes to their parents and the minimum covering radius are recomputed.

To get better understanding to the M-tree insertion process, we have provided a simple example, in this example we have inserted simple filters of Table 1 one by one to an empty M-tree of M=3 degree and showed the main changes in the tree. In Figure 4 first step (a), we have inserted filter A, B, and C into an empty M-tree, second step (b). We have inserted filter D into the M-tree, while the M-tree has only one full node (the root), we have to partition this tree node into a pair of nodes as described above. In the third step (c), we have inserted filter E, and F into the tree by finding the must suitable nodes that can cover them. Fourth step (d), we have inserted node G into root entry B, that required to partition the old node into two new nodes and link them to new entries in the root while there are more room to new entries in the root and then we have inserted node H into the must suitable node that can cover it with its covering area. Note that, in case of no node has covering area can cover the new node, we need to expand the nearest node covering area and considering that it is the must suitable insertion node. For instance, inserting node H into node C covering area required to expanding node C covering area while it is the nearest suitable node to node H and partitioning node B required to reducing node B covering area. Algorithm 1 presents the insertion process of the proposed system.

```
Algorithm 1: Proposed CM-tree Insert algorithm
Input: new filter r, root node RN;
Read SIP and DIP addresses;
SIP.R ←    Change Prefix To Range SIP);
DIP.R ← Change Prefix To Rang (DIP);
SIP.L ←    SIP:R upperLimit- SIP:R lowerLimit ;
DIP.L ← DIP: R upperLimit- DIP: R lowerLimit  ;

r.radius √((SIP.L)^2+(DIP.L)^2) / 2

r.hypersphere ← Create.hypersphere(SIP.R, DIP.R, r.radius);
r.Finger_print ← Create_Finger_Print(r.remaining_Fields);
r.B_Locations ← Create_Bucket_Locations(r.remaining _Fields);
 r ← RN;
Traverse the M-tree from root to the leaves;
At each level chose the most suitable node-entry that has covering area can cover
r.hypersphere;
if the chosen leaf K covering area can cover r.hypersphere and has a room for new entry
then

   Insert r into K;
   calculate r. distanceToParent;
   Update all the filterr in the path from K to RN;
Else
   If K is already full then
     Let  ε the set of the old entries of K plus r;
     Set two entries e1, e2 from ε so that the distance between them is the maximum among
     all other pairs;
     Create two new nodes Node1, Node2;
     Assign the remaining entries of ε to Node1, Node2 according to of which of Node1,
     Node2 can cover with minimum radius;
     Update covering area to Node1, Node2;
     Update distanceToParent to all of Node1, Node2 entries;
   If K cannot cover r then
     Expand K.radius inorder to cover r;
For entries of nodes that are in the path from Node1, Node2 to RN do
   Update (covering area) to cover Node1, Node2;
   Perform splits at the upper levels if necessary;
   If RN has to be split then
     Create new root;
     Tree height Tree height+1;
End Algorithm 1
```

2.3. Cuckoo filter programming

After the complete construction of the M-tree, the remaining fields of each set of filters that have the same parent (routing-node entry) have been hashed into CF and conjoined to the corresponding nodes. The leaves consist the link to the filters themselves and there is no need to integrate CFs into leaves. Each of the routing-node entries will has CF hashed the remaining field of all of its children. The routing-node entries of the upper level will contain a bigger CF contain all the content of its children CFs. The murmur hashing function [25] have been modified and used to hash the remaining field into CFs so that all remaining fields of

one filter are used together to generate one fingerprint, and a customized CRC32 has been used to generate bucket location.

The generated fingerprint has the shape of a connected long fingerprint but in fact it is generated as pieces of fingerprints, a small fixed-size fingerprint for each field, and then these fingerprints are combined to be a continuous fingerprint to represent all the remaining fields. A locality-sensitive hashing function [26] has been used to generate a locality-sensitive fingerprints piece for the SP, and DP fields (range fields). A fixed set of zeros was used as a fingerprint piece to represent the wild-card fields. The type of any field determines the length of its footprint piece. The fingerprint pieces for all the remaining fields are assembled in a specific order to produce the final fingerprint. Due to the dependence of the length of the bucket locations filter-set CF size which is variable in the proposed tree, bucket locations of varying lengths were used in different tree levels. One CF has been integrate to each routing-node entry to represent all of its children remaining fields, and another CF has been integrates with each routing-node at the upper tree levels by merging all of its children CFs.

2.4. CM-tree query

To perform an optimal query on the proposed CM-tree looking for a match with the incoming packets, we will perform tree descend pruning nodes whenever possible. Firstly, the SIP and DIP addresses have been extracted from each query packet and changed to numbers, and CF fingerprint and two buckets number have been generated. Next, the CM-tree are traversed up-down looking for routing nodes that its center e, radius r_e intersects the query packet with center q, radius r_q. Whenever a node is removed from the query queue, the query process is moved to its children. Before pruning any hyper-sphere n, we should prove that no node entry e below to n is closer to queried packet q than queried packet radius r_q.

The proposed CM-tree query process is presented in Algorithm 2, a hyper-sphere (Pkt. hypersphere) is generated for each incoming packet Pkt, and the CM-tree is queried up-down. At each non-leaves level, a CM-tree-query recursive call is made for the routing-node entries with covering area that cover queried packet radius if and only if it gets a positive remaining-fields query result for its integrated query filter. For the leaves level, a fill mapping is performed between the filter fields and the queried packet header fields, and add the matched filters to the answer set.

```
Algorithm 2: CM tree Query algorithm
Input: Query Packet Pkt, root node RN;
Read (SIP & DIP);
SIP.R ←    Change Prefix To Range (SIP);
DIP.R ← Change Prefix To Range (DIP);
SIP.L ←    SIP:R upperLimit- SIP:R lowerLimit ;
DIP.L ← DIP: R upperLimit- DIP: R lowerLimit

Pkt.radius
Pkt.hypersphere ←Create_hypersphere (SIP.R, DIP.R, Pkt.radius);
Pkt.Fingerprint← Create_Finger_Print (Pkt.remaining_Fields);
Pkt.B Locations ←Create_Bucket_Locations(Pkt.remaining_Fields);
Ptr  ← Root;
If Ptr is routing-node then
   while There routing-node entries k count in Ptr > 0 do
      If Ptr.k.coveringArea covers Pkt.hypersphere then
         Pkt.Finger_print Create Finger Print (Ptr.remaining_Fields);
         Pkt.B_Locations=Create Bucket_Locations(Ptr.remaining_Fields);
       If Cuckoo_F_Query(Ptr.k.CF,Ptr.Finger_print,Ptr.B_Locations)then
          Call CM_tree Query( Ptr,Ptr.k.child);
Else
    /* leaves */
    While node entries k in Ptr > 0 do
      If all Ptr.k filter fields match Pkt header fields then
         set Ptr.k as a results;
End Algorithm 2
```

3. RESULTS AND DISCUSSION

In this performance evaluation, four of the most relevant metrics including (filtering throughput, processing latency, memory requirements, and filtering accuracy) have been used to evaluate and compare the proposed method against a set of related modern methods. To perform this evaluation, three types of filter-set have been used including FW, IP chain (IPC), and access control list (ACL) with the size of (100 filters, 1 K filters, 10 K filters, 100 K filters, and 1 M filters). A Classbench tool [27] has been used to generate these filter-sets. We have performed this evaluation on a platform with an Intel Core i7-Q720

processor, with a CPU @ 1.60 GHz 4 and 8 GB DDR3 RAM. A set of related modern method including (BSOL-RC [19], decomposition-based approach for scalable many-field packet classification on multi-core processors (DBAMCP) [20], and BF-AQT [21]) have been chosen to compare the proposed method results. We have re-coded and implemented these methods in our platform and on our filter-sets to get a fairer comparison. All the parameters of the compared methods have been set to the optimum parameters. M-tree degree has been set to K=6, CF buckets have been set to b-internal=1,024 bucket with six slots, and maximum allowed reallocation=130, the size of bloom filter in the BF-AQT has been set to the same total size of our CF, and the optimum bucket size (4 slots) for BSOL-RC [27].

3.1. IP packet filtering throughput

The IP packet filtering throughput is the average number of the packet that our proposed method can filter in a single second. From Figure 5, we can see that the proposed classifier can classify up to 11 MPpS and 8.95 MPpS as average. The filtering throughput rate is gradually decreasing very slowly as the size of the filter set increases due to the increase in size of CM-tree and its integrated filters that require an increase in the required memory access rate. The proposed method clearly outperforms the other method [19]–[21]. With the small filter-sets, other trees perform well due to its structure that prevents split nodes if that split leads to the production of overlapped nodes, This structure is the same reason that makes it plumper with the large filter-sets which reduces its throughput. It clear that our method outperforms these methods in up to 2X in some cases. BSOL-RC [17] builds a decision trie with very high level so that it needs a separated decision trie for each field in the filter and a new trie level when ever the fields ranges are overlapped. DBAMCP [20] performs a sequential search to its integrated bloom filters and hash tables, also it need many hash tables that affects its search time. It can be noticed that using CF gives the proposed method an advantage over the DBAMCP [20], because it uses bloom filter, which is known to be slower than CF. BF-AQT [19] needs more processing time to run its merging algorithm, so that BF-AQT works on each field individually to get partial results, and perform merging process to calculate the final result.

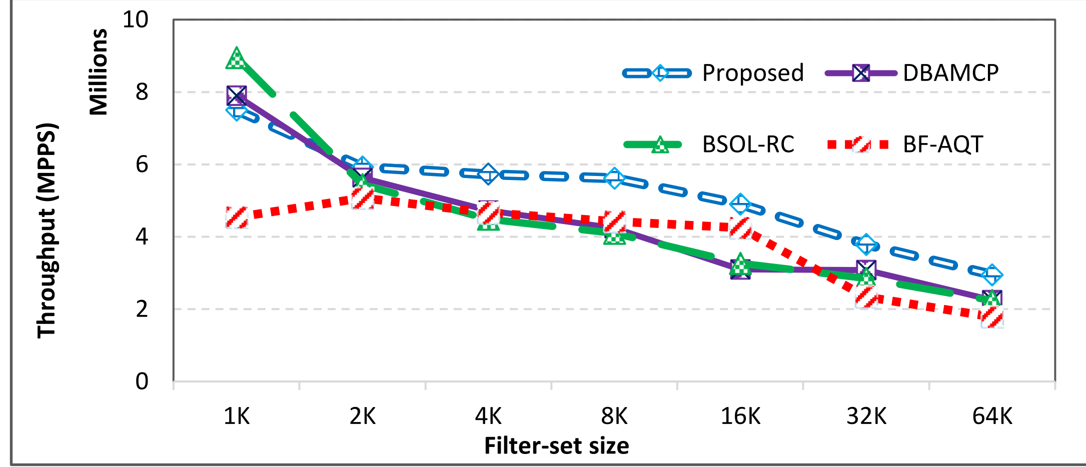


Figure 5. Average throughput comparison using rule sets of variant sizes against BSOL-RC [17], DBAMCP [20], and BF-AQT [19]

3.2. Memory requirements

In memory requirements metric, the entire memory required to store our data structure has been calculated and divided by the number of filters in our filter-set. We have defined byte per filter (BpF) as the average number of bytes that required to store each filter in our data structure. As can be seen from Figure 6, to store filter-sets of size (5-100) K filter, our proposed approach needs (48-112) byte, and only 92 byte in average. This allows it to support high scalability in term of the required memory. CM-tree stores more information at building time in order to restore the expensive calculating query time, such information need more storage memory. Compared to the rest of method [19]–[21], the proposed method provides better performance in almost all cases in this metric.

3.3. Processing latency

The processing latency has been defined as the average time our method needed to filter one packet measured in ms, we used filter-sets of size (1 K-1 M). The results were compared against the method of

[19]–[21]. The result of the comparison, as we see in the Figure 7, show that the proposed method outperforms all of the compared methods in terms of the processing latency except in some cases, especially if the filter-sets are small, where the performance of the most compared methods converges.

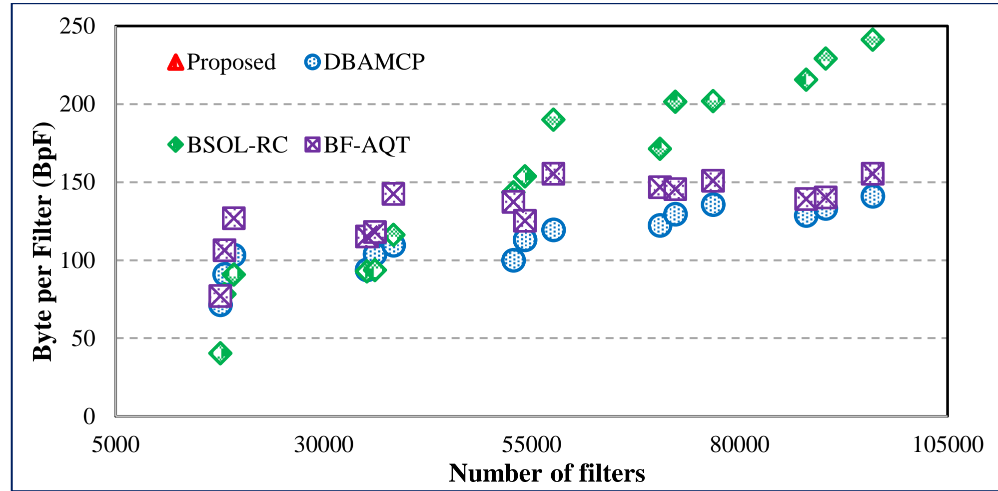


Figure 6. Required bytes per filter (BpR) comparison using rule sets of variant sizes against BSOL-RC [17], DBAMCP [19], and BF-AQT [20]

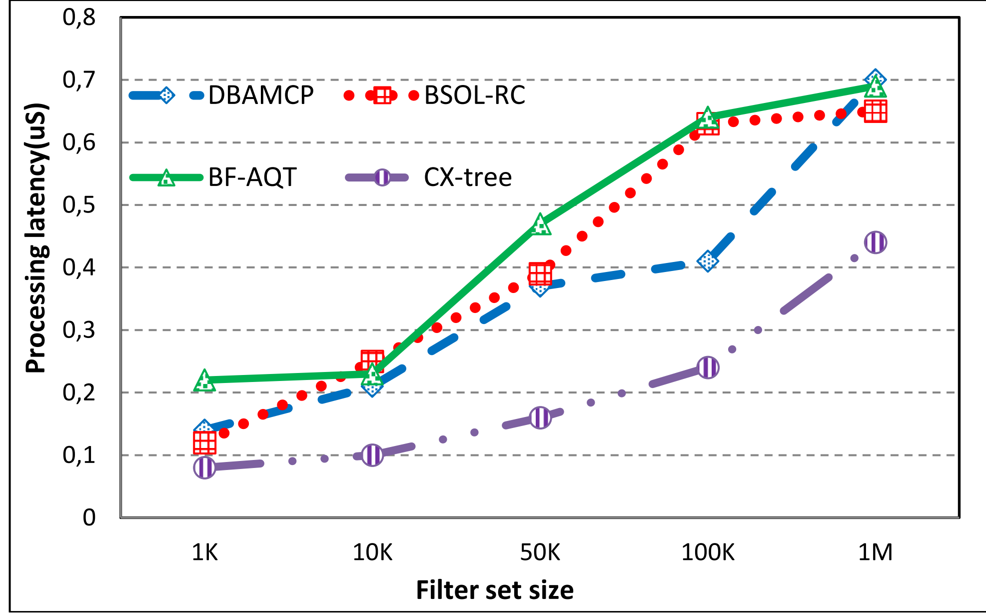


Figure 7. Comparison in terms of average processing latency against BSOL-RC [19], DBAMCP [20], and BF-AQT [21]

3.4. Classification accuracy

It is practically known that the results derived from the use of data structures with approximate results such as CF accept a percentage of error. This percentage depends on several factors including the size of the filter, the amount of data stored in it, and the type of hashing functions used to indicate the location and fingerprint of each component. Therefore, we decided to measure the rate of occurrence of such errors in the proposed structure. the false-positive rate has been calculated by classifying up to Million packets that do not belong to any class and measuring the number of times the proposed method was missed and classified them for a specific class. The results have been depicted in Table 2, and it is clear that the false positive rate

is only (1.2∗10−4) and this percentage is considered very small and such quantities of errors do not affect the integrity of the classification. On the other hand, the false-positive rate was compared against the other compared methods, and the results showed that the proposed method has the lowest false-positive rate among the rest of the methods due to using the CF instead of the bloom filters. Also, in general, the other methods have a relatively acceptable false-positive rate, if we take into account that the worst false-positive rate among the three methods is (3.34∗10−4).

Table 2. Result of the proposed CM-tree comparison in terms of false positive rate against BSOL-RC [19], DBAMCP [20], and BF-AQT [21] with variants size of filter-sets

	Tracing data				False_positive probability
	1 (K)	10 (K)	100 (K)	1 (M)	
CM-tree	0	1	12	115	0.000123
BSOL-RC	1	7	47	316	0.000334
DBAMCP	0	3	21	189	0.000192
BF-AQT	1	5	36	295	0.000303

4. CONCLUSION

In general, packet filtering approaches that belong to the category of the geometric algorithm have either geometric storage complexity or a polylogarithmic number of memory accesses. The M-tree has been used in the proposed method. It has preference over the rest of tree data structures in terms of the range query because its structure allows storing important data at the preprocessing time instead of extracting it at the search time. At the same time, M-tree like other tree data structures requires a relatively large amount of storage. Therefore, we have integrated the CF into its nodes. In practical, CF requires O(1) query time, as it helps to improve the performance of the M-tree in terms of the required storage so that it stores fingerprints instead of storing the items themselves. Moreover, it eliminates many unnecessary searches before the search reaches the leaves level. The results indicated that the proposed data structure has better storage requirements compared to other methods in addition to its strength in terms of other evaluation aspects.

REFERENCES

[1] N. Sundareswaran and S. Sasirekha, "Packet filtering mechanism to defend against DDoS attack in blockchain network," in *Evolutionary Computing and Mobile Sustainable Networks*, 2022, pp. 201–214. doi: 10.1007/978-981-16-9605-3_14.

[2] D. Salopek, "Hybrid hardware/software datapath for near real-time reconfigurable high-speed packet filtering." M.S. thesis, Fac. of Elect. Eng. and Comput., University of Zagreb, Zagreb, Croatia, 2022.

[3] C. L. Lee, G. Y. Lin, and Y. C. Chen, "Fast conflict detection for multi-dimensional packet filters," *Algorithms*, vol. 15, no. 8, pp. 1–17, Aug. 2022, doi: 10.3390/a15080285.

[4] W. Meng, W. Li, and L. F. Kwok, "Towards effective trust-based packet filtering in collaborative network environments," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 233–245, Mar. 2017, doi: 10.1109/TNSM.2017.2664893.

[5] H. Lim, N. Lee, G. Jin, J. Lee, Y. Choi, and C. Yim, "Boundary cutting for packet classification," *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, pp. 443–456, Apr. 2014, doi: 10.1109/TNET.2013.2254124.

[6] T. Ganegedara, W. Jiang, and V. K. Prasanna, "A scalable and modular architecture for high-performance packet classification," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1135–1144, May 2014, doi: 10.1109/TPDS.2013.261.

[7] Open Networking Foundation, "OpenFlow switch specification: Version 1.5.1 (protocol version 0x06)," *The Open Networking Foundation*, 2015. https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf (accessed Apr. 03, 2022).

[8] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, Dec. 2014, pp. 75–88. doi: 10.1145/2674005.2674994.

[9] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *Proceedings of the 23rd International Conference on Very Large Databases, VLDB 1997*, 1997, pp. 426–435.

[10] M. Al-hisnawi and M. Ahmadi, "QCF for deep packet inspection," *IET Networks*, vol. 7, no. 5, pp. 346–352, Sep. 2018, doi: 10.1049/iet-net.2017.0037.

[11] A. X. Liu, C. R. Meiners, and E. Torng, "Packet classification using binary content addressable memory," *IEEE/ACM Transactions on Networking*, vol. 24, no. 3, pp. 1295–1307, Jun. 2016, doi: 10.1109/TNET.2016.2533613.

[12] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," in *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication - SIGCOMM '98*, 1998, pp. 203–214. doi: 10.1145/285237.285283.

[13] N. K. Sreelaja and G. A. V. Pai, "Ant colony optimization based approach for efficient packet filtering in firewall," *Applied Soft Computing*, vol. 10, no. 4, pp. 1222–1236, Sep. 2010, doi: 10.1016/j.asoc.2010.03.009.

[14] H. Salehi, H. Shirazi, and R. A. Moghadam, "Increasing overall network security by integrating signature-based NIDS with packet filtering firewall," in *2009 International Joint Conference on Artificial Intelligence*, Apr. 2009, pp. 357–362. doi: 10.1109/JCAI.2009.12.

[15] C. Manusankar, S. Karthik, and T. Rajendran, "Intrusion detection system with packet filtering for IP spoofing," in *Proceedings of 2010 International Conference on Communication and Computational Intelligence, INCOCCI-2010*, 2010, pp. 563–567.

[16] A. A. Abdulhassan and M. Ahmadi, "Cuckoo filter-based many-field packet classification using X-tree," *The Journal of Supercomputing*, vol. 75, no. 9, pp. 5667–5687, Sep. 2019, doi: 10.1007/s11227-019-02818-5.

[17] H. Lu and S. Sahni, "O(log W) multidimensional packet classification," *IEEE/ACM Transactions on Networking*, vol. 15, no. 2,

pp. 462–472, Apr. 2007, doi: 10.1109/TNET.2007.892845.
[18] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 4, pp. 25–36, Oct. 1997, doi: 10.1145/263109.263136.
[19] Y.-C. Cheng and P.-C. Wang, "Packet classification using dynamically generated decision trees," *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 582–586, Feb. 2015, doi: 10.1109/TC.2013.227.
[20] H. Lim and H. Y. Byun, "Packet classification using a bloom filter in a leaf-pushing area-based quad-trie," in *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, May 2015, pp. 183–184. doi: 10.1109/ANCS.2015.7110131.
[21] Y. R. Qu, S. Zhou, and V. K. Prasanna, "A decomposition-based approach for scalable many-field packet classification on multi-core processors," *International Journal of Parallel Programming*, vol. 43, no. 6, pp. 965–987, Dec. 2015, doi: 10.1007/s10766-014-0325-6.
[22] P. Warkhede, S. Suri, and G. Varghese, "Multiway range trees: Scalable IP lookup with fast updates," *Computer Networks*, vol. 44, no. 3, pp. 289–303, Feb. 2004, doi: 10.1016/j.comnet.2003.09.004.
[23] R. Pagh and F. F. Rodler, "Cuckoo hashing," *Journal of Algorithms*, vol. 51, no. 2, pp. 122–144, May 2004, doi: 10.1016/j.jalgor.2003.12.002.
[24] S. Guhlemann, U. Petersohn, and K. Meyer-Wegener, "Reducing the distance calculations when searching an M-Tree," Datenbank-Spektrum, vol. 17, no. 2, pp. 155–167, Jul. 2017, doi: 10.1007/s13222-017-0258-5.
[25] L. Chi, and Z. Xingquan, "Hashing techniques: A survey and taxonomy," *ACM Computing Surveys (CSUR),* vol. 50, no. 1, pp. 1-36. 2017, doi: 10.1145/3047307.
[26] L. Paulevé, H. Jégou, and L. Amsaleg, "Locality sensitive hashing: A comparison of hash function types and querying mechanisms," Pattern Recognition Letters, vol. 31, no. 11, pp. 1348–1358, Aug. 2010, doi: 10.1016/j.patrec.2010.04.004.
[27] D. E. Taylor and J. S. Turner, "ClassBench: A packet classification benchmark," *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, pp. 499–511, Jun. 2007, doi: 10.1109/TNET.2007.893156.

BIOGRAPHIES OF AUTHORS

Aladdin Abdulhassan received the BS and MSc in computer science from University of Babylon in Iraq in 2005 and 2011, respectively. From 2011 to 2014, he worked as a lecturer in the Faculty of Sciences at the Department of Computer Science in the University of Babylon. In September 2014, he enrolled in the Faculty of Engineering at the Department of Computer Engineering and Information Technology, University of Razi, Kermanshah, Iran as a full time Ph.D student. At January 2019, received his Ph.D. At 2019, he started working as a lecturer in the Faculty of Information Technology at the Department of Information Network in the same university, in addition to managing the International Ranking Division at the university and participating in the discussion of many master's theses, as well as the administrative committees. His research interest includes information security, SDN, packet classification, software-defined networking, and network processing. He can be contacted at email: aladdin.alsharifi@uobabylon.edu.iq.

Roaa Shubbar received the B.S. degree in Computer engineering from University of Nahrain, Baghdad, Iraq in 2004. She received the M.Sc. degrees in Computer engineering from University of Nahrain, Baghdad, Iraq in 2007. From 2012 to 2014 she worked as a lecturer in Faculty of Information Technology, Department of Software, University of Babylon, Babylon, Iraq. In 2018, she completed her PhD from Razi University, Iran. Her research interests include computer communications, network processing. She can be contacted at email: roaa.shubber@uobabylon.edu.iq.

Mohammad Alhisnawi received the B.S. degree in Computer science from University of Babylon, Babylon, Iraq in 2004. He received the M.Sc. degrees in Computer science from University of Babylon, Babylon, Iraq in 2010. From 2004 to 2014 he worked as a lecturer in Faculty of Information Technology, Department of Information Networks, University of Babylon, Babylon, Iraq. In 2018, he completed his PhD from Razi university-Iran. His research interests include computer security, network processing. He can be contacted at email: mohammad.alhisnawi@uobabylon.edu.iq.