I.J. Information Technology and Computer Science, **2021, 5, 51-63**
Published Online October 2021 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2021.05.05

An Enhanced List Based Packet Classifier for Performance Isolation in Internet Protocol Storage Area Networks

Joseph Kithinji
Meru University of Science and Technology, Meru, Kenya
E-mail: joskithinji2014@gmail.com

Makau S. Mutua and Gitonga D. Mwathi
Department of Computer Science, Meru University of Science and Technology, Meru, Kenya
Department of Computer Science, Chuka University, Chuka Kenya
E-mail: {smutua@must.ac.ke, dgmwathi@chuka.ac.ke}

Received: 31 July 2021; Revised: 10 August 2021; Accepted: 23 August 2021; Published: 08 October 2021

Abstract: Consolidation of storage into IP SANs (Internet protocol storage area network) has led to a combination of multiple workloads of varying demands and importance. To ensure that users get their Service level objective (SLO) a technique for isolating workloads is required. Solutions that exist include cache partitioning and throttling of workloads. However, all these techniques require workloads to be classified in order to be isolated. Previous works on performance isolation overlooked the classification process as a source of overhead in implementing performance isolation. However, it's known that linear search based classifiers search linearly for rules that match packets in order to classify flows which results in delays among other problems especially when rules are many. This paper looks at the various limitation of list based classifiers. In addition, the paper proposes a technique that includes rule sorting, rule partitioning and building a tree rule firewall to reduce the cost of matching packets to rules during classification. Experiments were used to evaluate the proposed solution against the existing solutions and proved that the linear search based classification process could result in performance degradation if not optimized. The results of the experiments showed that the proposed solution when implemented would considerably reduce the time required for matching packets to their classes during classification as evident in the throughput and latency experienced.

Index Terms: Performance Isolation, Storage Area Network, Throttling, Optimization, Metrics.

1. Introduction

Consolidation of storage into storage area network (SAN) rather than using separate physical infrastructure is a common approach meant to save on hardware and operational costs. However, this results into resource contention among the network users as they only get a fraction of the resource shared[1,2] such as disk cache and bandwidth. Contrastingly, the traditional free for all option is neither a solution as it may end up in significant efficiency degradation when the network is shared by multiple users [3,1,4]. To alleviate this challenge, performance isolation has been proposed as the appropriate technique for reducing interference among the network users [4,5].

Consequently, various tools have been developed to provide for performance isolation using either throttling or cache partitioning. Façade, Parda ,Triage and Pclock[6,7,8, 9] which implements performance isolation by throttling individual I/O requests from multiple clients to ensure storage devices do not saturate. Argon[11] uses cache partitioning and quanta based disk time scheduling to implement performance isolation between workloads.

Enforcing performance isolation either using throttling technique or cache partitioning requires packets be classified in order to be isolated[12]. This entails association of packets to classes based on the packet header information[12,13]. However, all the above named solutions assume that the classification process does not affect overall performance of the storage area network when used in implementing performance isolation[15]. We hypothesize that this assumption may not be true given that all the performance isolation techniques discussed are dynamic and keep on changing with traffic patterns[12,15]. In addition, in the above mentioned techniques the authors employed linear search when matching packets to classification rules. Linear classifiers are known to have problems including bigger rule problem, redundancies, shadowed rules, delays due to sequential computation and swapping rules problem[16]. All these problems result in the change of classification policy and delays if not addressed.

For a classification policy consisting of a list of n unsorted rules $r_1, r_2, ..., r_n$ a packet d_i is said to match rule r_i if the fields of rule r_i match the header field of packet d_i. A packet d_i may match any of the rules r_i, $i = 0,1,2,...,n-1$. If the matching rule is found on the i^{th} position, then $i+1$ comparisons will have been made. Thus, the average number of comparisons for a successful search denoted as *C* is as illustrated in (1).

$$C = \sum_{i}^{n-1} (i+1) \tag{1}$$

The optimization problem is to arrive at a legitimate rule order that results in the optimal cost *C*.

This paper puts forward a technique that improves on delays experienced when matching packets to rules when using linear search based classifiers. The study employs the techniques of sorting rules based on hit ratio, partitioning rules, jump search and using the portioned rules to build linear tree rule structure in order to eliminate the problems associated with linear search classification. Finally, the paper validates the optimization of the classifier using time complexity, response time and accuracy metrics while using the classes generated to bind resources to users in order to provide performance isolation. Based on the throughput and response time of the proposed solution, empirical results indicate that lack of optimization in classification would lead to considerable degradation of performance in a SAN while implementing performance isolation.

2. Related Works

Various techniques have been proposed to optimize the performance of list-based classifiers. In their work, Hamed and others in [17] proposed a technique for optimizing firewall filtering rules by calculating the traffic statistics then using the results to dynamically reorder firewall rules. When implemented the solution proved to be simple and light weight. However, its early rejection property may cause more packets to be dropped which would be detrimental to overall QOS especially for storage area networks. The paper uses the dynamic reordering of classification rules based on priority which is calculated from hit ration of rules.

El-Atawy, Samak, Al-Shaer, & Hong in [19] proposed two methods namely segment based tree search and segment based list search. The segment based tree search uses Huffman trees and traffic characteristics for each segment to reduce the search time. However the technique was proven to have a lot of overhead especially when it comes to maintaining the tree[20]. To eliminate this overheard, they [21] used the segment based list search which included a most recently used list which is placed at the top of the classification rule list to reduce the search time. However, it was observed that this method is more useful when the traffic is steady [19].In the paper jump search which is similar to segment based search is used to reduce overall search time.

Trabelsi & Zeidan in [23] proposed a method which rejects packets early and also accepts packets as early as possible. The early acceptance is achieved with the use of splay trees decisions which are updated with networking history of the traffic characteristics to ensure more frequent packets are processed faster. For rejection a multilevel approach for filtering packets is used before the decision for rejection is made. This is done in an attempt to ensure that users are not denied service[19].In the proposed technique all packets are compared to all rules belonging to all segments to ensure fewer packets are dropped. In the paper this is achieved by eliminating the problems of shadowed rules and bigger rules. Shadowed rules are removed while bigger rules are put at the end of the list.

Named and Al-Shaer[24] used a branch and bound technique to resolve the optimal rule ordering problem by ensuring that the minimum number of rules are matched to the packets as well as maintaining the relationship among the rules. However, Vasu & Ganesh [25] observe that the proposed approach has linear space complexity and the resulting time complexity was proven to be polynomial. To reduce on time complexity the paper implements a linear tree structure. To reduce on the time taken for matching rules the paper adopts jump search as well as linear tree structure.

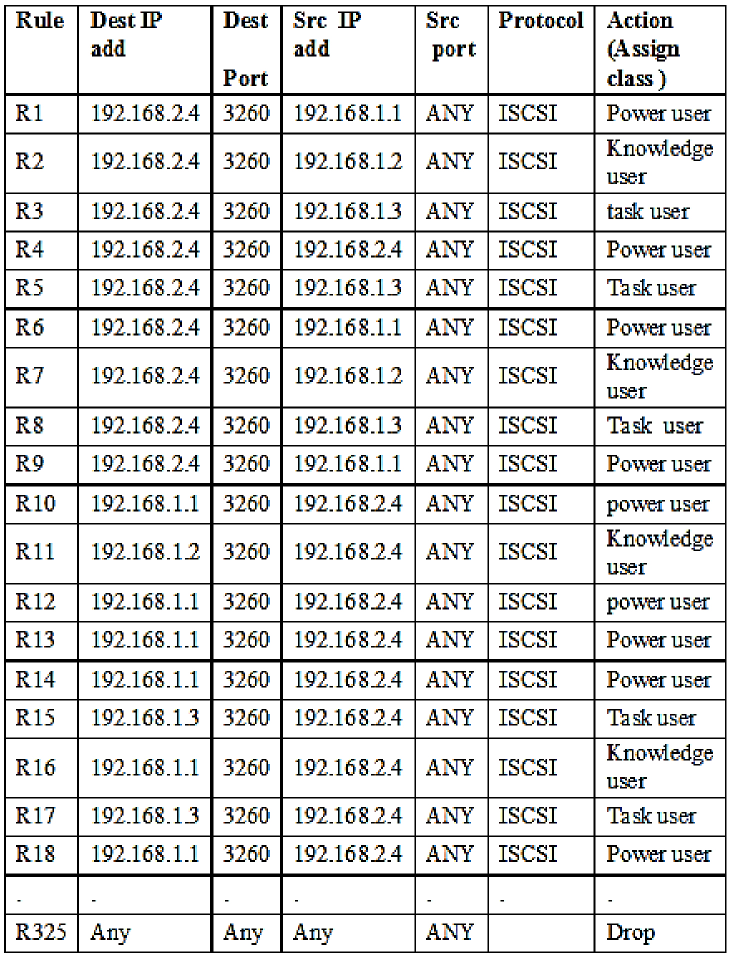
To reduce on the observed complexity, Vasu and Ganesh [19] proposed a technique for reordering packets based on the current network statistics and splitting the packets into *P* partitions to reduce the search time. The window size is used to store the history of the traffic pattern. To improve on this solution, we partition the rules instead of the traffic to reduce the match time since the overhead incurred on segmenting packets is more than that of segmenting rules. This follows from the fact that even in a medium sized organization the network could generate millions of packets unlike number of rules which could be far much less[24,25].

All the above approaches indicate the need and lack of integration of optimization techniques to achieve minimum number of matches for a classification action to be performed. Consequently, the proposed approach differs with the discussed approaches in its integration of rule ordering, rule splitting, use of jump search and use of linear tree rule structure to optimize the classification process. In addition, the proposed technique will be implemented in an IP SAN unlike previous solutions.

2.1. Limitations of Linear Search Based Classifier

Linear search based classifiers experience a number of limitations ranging from shadowed rule(s) to redundant rules. In order to put these limitations to perspective, we use Table 1 which illustrates a list of rules for a typical linear search based classifier policy.

Table 1. Sample List Based Classification Policy



Rule	Dest IP add	Dest Port	Src IP add	Src port	Protocol	Action (Assign class)
R1	192.168.2.4	3260	192.168.1.1	ANY	ISCSI	Power user
R2	192.168.2.4	3260	192.168.1.2	ANY	ISCSI	Knowledge user
R3	192.168.2.4	3260	192.168.1.3	ANY	ISCSI	task user
R4	192.168.2.4	3260	192.168.2.4	ANY	ISCSI	Power user
R5	192.168.2.4	3260	192.168.1.3	ANY	ISCSI	Task user
R6	192.168.2.4	3260	192.168.1.1	ANY	ISCSI	Power user
R7	192.168.2.4	3260	192.168.1.2	ANY	ISCSI	Knowledge user
R8	192.168.2.4	3260	192.168.1.3	ANY	ISCSI	Task user
R9	192.168.2.4	3260	192.168.1.1	ANY	ISCSI	Power user
R10	192.168.1.1	3260	192.168.2.4	ANY	ISCSI	power user
R11	192.168.1.2	3260	192.168.2.4	ANY	ISCSI	Knowledge user
R12	192.168.1.1	3260	192.168.2.4	ANY	ISCSI	power user
R13	192.168.1.1	3260	192.168.2.4	ANY	ISCSI	Power user
R14	192.168.1.1	3260	192.168.2.4	ANY	ISCSI	Power user
R15	192.168.1.3	3260	192.168.2.4	ANY	ISCSI	Task user
R16	192.168.1.1	3260	192.168.2.4	ANY	ISCSI	Knowledge user
R17	192.168.1.3	3260	192.168.2.4	ANY	ISCSI	Task user
R18	192.168.1.1	3260	192.168.2.4	ANY	ISCSI	Power user
.	.	.	.	.	.	.
R325	Any	Any	Any	ANY		Drop

A. Shadowed Rule

A shadowed is a rule that won't get to match since a rule preceding it will have matched all its packets[21]. In Table 1 R12 is shadowed by R1. Shadowed rules may bring about speed problems as well as security issues[28]. Shadowed rules therefore can be deleted without changing the classification policy[32,33,34].

B. Limitation about Swapping Position between Rules

Swapping rules can change the classification policy if the swapped rules result in putting packets in different classes that can be able to match the same packet. Changing the packet action would alter the accuracy of the classifier[32]. In Table 1 swapping R325 with any other rule would result in a different action[36,37].

C. Limitation on Redundant Rules

A redundant rule is one that has been implied by another one below it[32].In Table 1 R14 is redundant to R10.Redundant rules result in reduction in the speed of processing packets which waste classifiers processing time[38,39,40].

D. Bigger Rule Problem

A big rule is one that matches all packets. In Table 1 R325 is the bigger rule. In other words, the default rule. If bigger rule is placed before other rules, it shadows them. This brings about a design problem since the rules position is of significant importance[41,21].

E. Sequential Computation Limitation

In a listed based firewall the computation for packet classification is sequential which brings about speed problems when rules are many[26]. The time required for packet classification will increase with an increase in the number of rules[21].

2.2. Definition of User Classes and their Operational Metrics

The information technology industry classifies storage users as either task users, knowledge users or power users. Table 2 illustrates the operational resources required per user for the corresponding class and their associated limits in terms of disk space and IOPS.

Table 2. Operational Metrics per Class of User

Class of user	Description	Disk space	IOPS
Task user	Perform repetitive tasks Such as creating Simple documents	25 GB Disk space	5 IOPS
Knowledge user	Create complex Documents such As spreadsheets	40 GB Disk space	10-20 IOPS
Power user	Use CPU and graphics Intensive applications	40 GB Disk space	25 IOPS

Metrics for measuring storage performance include throughput in IOPS/KBs, latency and response time[39]. Block size is a unit of data that is read during and I/O operation[40]. The block size impacts throughput[42,43].Throughput is a product of IOPs and block size. Their relationship is as illustrated in the equation (2) [43,44].

$$Throughput = IOPs \times BlockSize \tag{2}$$

Queue depth is the number of I/O commands that can be queued at a time on a storage controller at the initiator side or at the target side. For small midsized storage area networks, a queue depth of 32 is recommended. Equation (3) is used to calculate the queue depth [43,44];

$$QueueDepth = IOPs \times \mathrm{Re}\,sponseTime \tag{3}$$

Storage level objective(SLO) is a quality of service aspect that can be used for measuring performance of a storage system or storage service provider[5]. A SLO is a combination of one or more QOS metrics with their corresponding values[4,42,44].

3. Methods and Materials

Various methods were used in this study as outlined in the in the following sections.

3.1. Methods

This study embarked on optimizing the process of packet matching during classification process for linear search based classifier. The methods of sorting the rule list, partitioning the rule list, jump search and building a linear tree rule structure to optimize the classification process.

To begin with, the system to be emulated was modelled for comparison purposes. To obtain the standard SLO requirements for each class of users, we used Table 2 and equation (3) and (4) to derive the SLO for classes of users based on the IOPs, block size and queue depth. The values for the SLO are throughput in Kb/s followed by IOPS and then response time. For a block size of 4kb the SLO for task, knowledge and power users is as follows; task users(20kb/s,5IOPS,6.4ms), Knowledge users(60kb/s,15IOPS,1.6-3.2ms) and power users(100kb/s,25IOPS,1.3ms). The same case applies for 64kb and 1 Mb block sizes.

A. Rules Priority Estimation and Sorting

Rule hits were established in order to determine the priority and order of the rules. Rules with highest number of hits are to be placed at the top t reduce the search time. The hits were to determine the positioning of rules for optimal performance[35,42,43].After the establishment of hits distribution then the sorting Algorithm 1 was applied.Algorithm 1 takes an array of rules and priorities as input then returns a sorted array or rules based on priority. For optimal performance rules with the greatest hits are placed at the top and those with fewer hits follow[2,47,48].This is expected to reduce the time taken for searching since the most commonly hit rules will be at the top. Therefore, most hits will be made without traversing the whole list. The formula for calculating priority is as illustrated in Equation(4).

The priority of the rules was established using the following equation

$$P_i = \frac{h_i}{N} \tag{4}$$

Where h_i the hit is count of rule i and N is the total number of rules.

An experiment was performed to determine the rule hits ratio and the results are as illustrated in Fig.3.

```
Algorithim1: Sorting algorithm
Input: An array of rules and their priorities
Output: sorted array of rules and their priorities
    1. For ($i = 1; i < n; i++$)
    2. If ( $p_i > p_{i+1} AND i \cap i+1 = \varnothing$ then
    3.     $temp = r_i$
    4.     $r_i = r_{i+1}$
    5.     $r_{i+1} = temp$
    6. Endif
    7. End for
```

B. Partitioning the Rule List

To reduce the amount search time, we use jump search instead of linear search. In its simplest implementation the jump search algorithm operates with jumps the size of the square root of the number of items[4,49,50,51,52] see Equation (5).Partitioning the rule list reduces the number of rules in each portion which makes searching take less time. This so due to the fact that partitions with more hits are placed ahead of those with less hits. Therefore more matches are made without traversing all the partitions present.Algorithim 2 was used for the jump search.

Therefore, we have

$$m = \sqrt{N} \tag{5}$$

For Table 1 we have 324 rules therefore we have *m*=18.

```
Algorithm 2: Jump Search
    1. $m = \sqrt{N}$
    2. If $d_i.f_i == r_i.f_i$
    3. Perform action $i$
    4. Else
    5. If $d_i.f_i > r_i.f_i$ then
    6.     $n = n + m$
    7. go to step 2
```

```
Algorithm 3: Splitting the rules
Input: array consisting of priorities of rules,
Output: partitioned array
1.  $N = array.Length();$
2.  $size = \sqrt{N}$
3.  Function portion (array, size) {
4.  const dividearray = [];
5.  For ($i = 0; i < n; i++$) {
6.      $const\ tail = dividearray[dividearray.length - 1];$
7.  if ($!tail$ □ $last.length == size$)
8.  dividearray.push([array[$i$]]);
9.  } else {
10. tail.push(array[$i$]); //Else add the current element into the chunk }}
11. return dividearray;}
```

Algorithm 3 will take input as an array of priorities of the rules. The algorithm will then proceed to split the array into chunks of length size. Then the algorithm will return a nested chunk of arrays.

C. Linear Tree Rule Structure Design

The listed classification rules in Table 1 have been theoretically analyzed in section 2.2 and proven to have conflicts and redundant rules. To eliminate these problems we built a linear tree rule structure to be used by the classifier[53,19,54]. The tree rule structure was arrived in two steps that is sorting step and the partitioning step as illustrated in sections 3.2 and 3.3. With a tree rule structure the cost of sequential file computation is reduced from *O(N)* to O*(Log N)* using the proposed tree rule structure. To further improve on the design, we use range matches instead of exact matches as the root node.

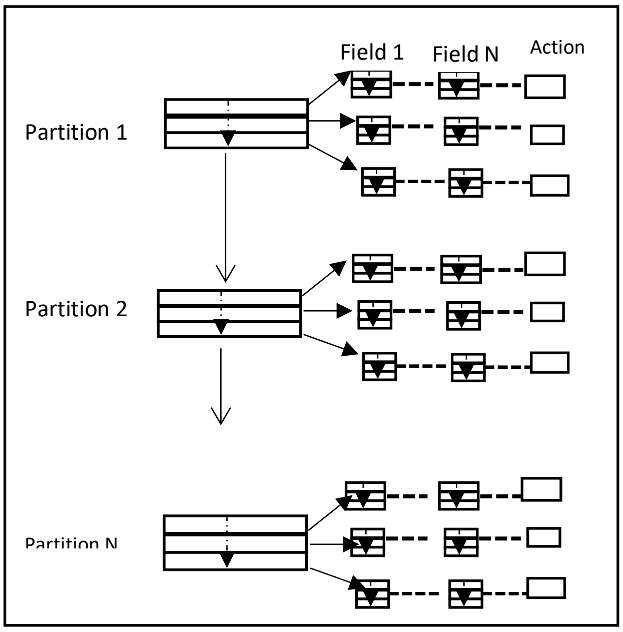


Fig.1. Linear Tree Structure Building

The root node contains as many lines as there are users in the network. Since we are focusing on allocation of resources we have specified the source IP address as the root node to guarantee resources to packets as they traverse the network from source to destination[32].

The linear search optimization technique is integrated in the design of the classifier. Fig.1 illustrates how the tree rule was built while Fig.2 illustrates the result of building the tree rule.

```
Algorithm 3: Optimized Classification Algorithm
We define the following for the algorithm
N-Total number of rules.
$m = \sqrt{N}$ ; $m$ is the number of the partitions.
$n =$ Number of rules in each partition.
R-A list of rules. $R = \{r_1, r_2, .., r_n\}$
D- A list of packets. $D = \{d_1, d_2, ..., d_n\}$.
F-A set of packet header fields/column fields. $F = \{f_1, f_2, f_3, f_4, f_5\}$.
P-Partitions of rules. $P = \{p_1, p_2, ..., p_n\}$
A- A set of packet classes $A = \{a_1, a_2, ..., a_n\}$
W-default class.
Update()-a function that keeps track of recent traffic history.
Reorder () - a function that reorders rules based on traffic characteristics.
Resplit ()-a function that re-splits the reordered rules into partitions.
    1. INPUT; R, A, W
    2. OUTPUT;
    3. For($m = 0; m < n; m++$)
    4. For($i = 0; i < n; i++$)
```

```
5.  If   $d_i.f_i = r_i.f_i.p_i$ then
6.  Output $a_i$
7.  Else
8.  Output $w$
9.  Endif,Endfor ,Endfor
10. If( $m == n$ )
11. Update();
12. Reorder();
13. Resplit();
14. Endif
```

Algorithm 3 will take input as incoming packet d_i.The fields of the packet d_i is compared to the fields of rule r_i.If they match then the corresponding action is performed. If none of the rules match in the current block of rules then the counter m is incremented to move to the next block. If none of the rules match then the packets are placed in the default class w. When all the portioned blocks are searched then the update function is called for capturing current network changes in terms of hit ratio. Then again, the rules are reordered based on the new hits statistics using reorder () function. After reordering, the resplit() function is called which splits the rules into partitions which are used to build linear tree rule.

3.2. Materials

Parkdale, was the tool of our choice for generating traffic as it uses a queue depth of 32 which is the default for windows, given that the initiators run on windows. Parkdale is freeware tool developed by the SZ development and is used for measuring the performance of local, optical, external and network drives. For all the experiments a file size of 4GB which is the maximum file size possible with the Parkdale tool was simulated for reads and writes. To characterize system behavior during experiments different tools were used to collect the various aspects of the system behavior during execution.

Data was collected using Parkdale, wireshark and the *tc* statistics command. Parkdale was used to measure throughput and response time. The *tc –s qdisc dev eth 0* was used to show class statistics with information under each class. The *tc –s qdisc dev eth 0* displays the statistics of a particular class including number of packets sent, number of packets dropped and latency. The wireshark tool was used to collect information on hit ration as per the initiators IP address. Wireshark is an open source and cross platform tool used for network analysis. It was developed by the wireshark team.

For our experiments we used three computers (Intel 2.8 GHZ CPU with 2GB of RAM and 500 GB hard disk). Their roles were that of target, router and initiator respectively. The router contains two Ethernet cards of 100 Mbps and it was directly connected with the target and the initiator. The router is the computer sitting at the middle. To generate storage traffic, we used Parkdale.

4. Results and Discussion

In the following sections we outline the results based on time complexity analysis, hit ratio, performance evaluation and classifier accuracy.

4.1. Time complexity analysis

Time complexity for a listed rule in Table 1 is *O(N)*[55,56].Time complexity for sequential tree rule is *Log(N)* but since we have used IP address and port ranges the time complexity for the tree rule slightly increases to $1+Log(N)$.Where N is the number of rules.[35,57,58].

From Table.1 if we assume that the chances for a match is the same for each rule. Then the average number of matches is $(18/2)\times 5\times C = 30C$ where C the time is required to compare one field of a packet to one field of a rule. Five is the number of fields in the table that require comparison.

With the tree rule in Fig.2 we have on average 5 lines in the source IP field to get $(1+Log_5)C$,one line in the destination port field to get $(1+Log_1)C$,three lines in the destination IP field to get $(1+Log_3)C$, one line in the source port field $1+Log_1)C$ and one line in the in the protocol field $(1+Log_1)C$ [29].

Consequently, with the tree rule classifier we have $\{(1+Log_5)C+(1+Log_1)C+(1+Log_3)C+(1+Log_1)C+(1+Log_1)c = 6.1C$ .The mathematical simulation show that optimization strategy applied result in reducing the operational cost by 20%.

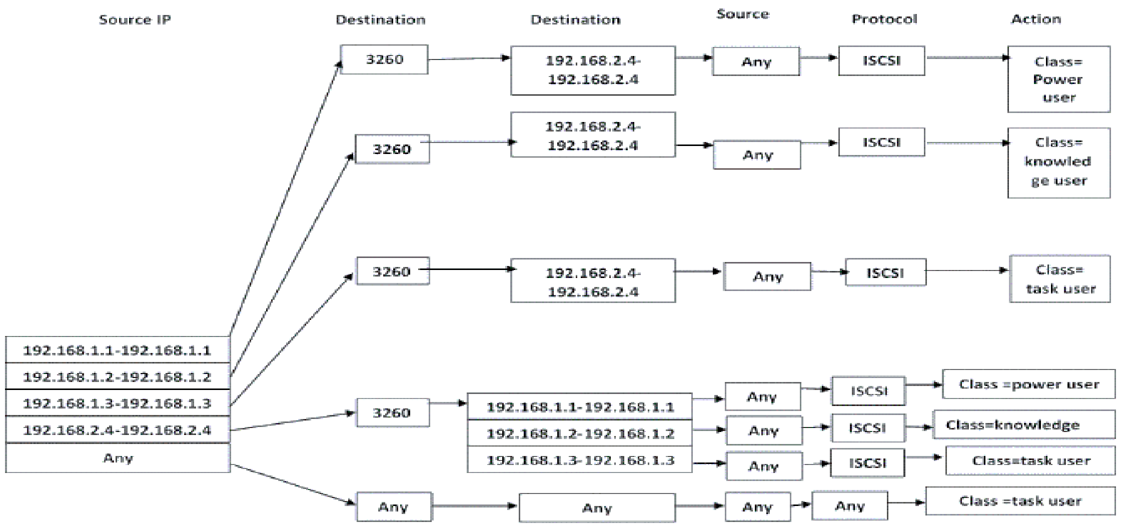


Fig.2. Linear tree rule structure

4.2. *Hit Ratio*

Fig.3 shows the distribution of the hit for rules over an attempt to read/write files of size 4GB with blocks of size 4KB, 64KB and 1MB.The results show that heavy hit rules are experienced from the power users followed by the knowledge users and lastly the task users. Therefore, to optimize performance, rules associated with power users are put at the top. Based on the rule hit distribution we apply the sorting algorithm[56].

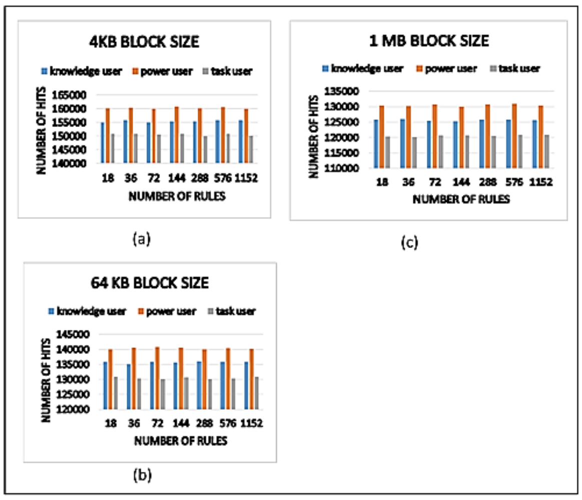


Fig.3. Rule hits distribution over varied block sizes

4.3. *Performance Evaluation*

Experiments were done to determine how well the proposed solution conforms to the SLO of various users. Our evaluation was composed of experiments that examine the following questions:

1. How throughput vary with changes in the number of rules and the block size?
2. How does response time vary with changes in the number of rules and the block size?

In Figs.4(a) and 5(a) we observe a steady throughput degradation for the list based classifier due to an increase in the number of rules. Initially when numbers of rules are less, the throughput of the list based classifier is similar to that of the proposed solution. As the number of rules increases the performance of the list based classifier deteriorates while that of the proposed system stabilizes. Another observation is that since the throughput is a product of block size the higher the block size the higher the throughput.

From Figs.4(a) and 5(a) we further observe that the storage users are not able achieve their SLO with the list based classifier, this is because the list based classifier causes delays due to the increase in the number of rules and therefore results in reduced performance as the rules increase.

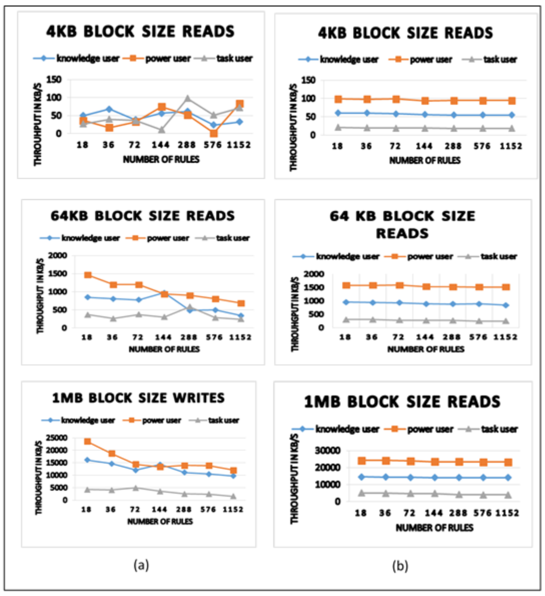


Fig.4. Writes throughput comparison (a) Without the proposed solution and (b) With the proposed solution for varied block sizes.

However, with the proposed solution as illustrated in Figs.4 (b) and 5(b), rule search time is reduced during the classification process and therefore all the classes of users are able to achieve an SLO close to the system being emulated. This is intuitively consistent with what we expect that the users should meet SLOs close to the system modelled in section 4 irrespective of the number of rules[29].

In addition these results are consistent with what we would expect and also with research done in [32] where experiments were performed to compare the performance of a IPtabels which is a list based firewall versus the optimized list based firewall. The results showed that the performance of the optimized list based firewall is not drastically affected by the increase in the number of rules unlike that of list based firewall[2].

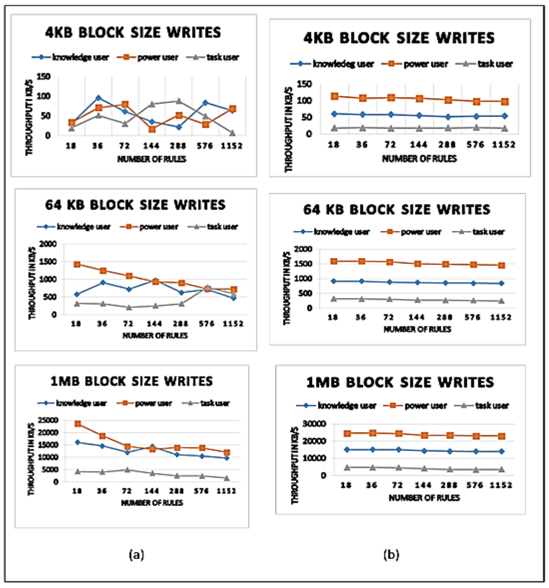


Fig.5. Writes throughput comparison (a) Without the proposed solution and (b) With the proposed solution for varied block sizes.

From Figs.6 (a) and 7(b) we observe that the response time for the list based classifier steadily increase with the number of rules. On the other hand the response time of the proposed classifier slightly increases then stabilizes[29]. Indicating that the performance of the proposed classifier is not adversely affected by the number of rules[2].

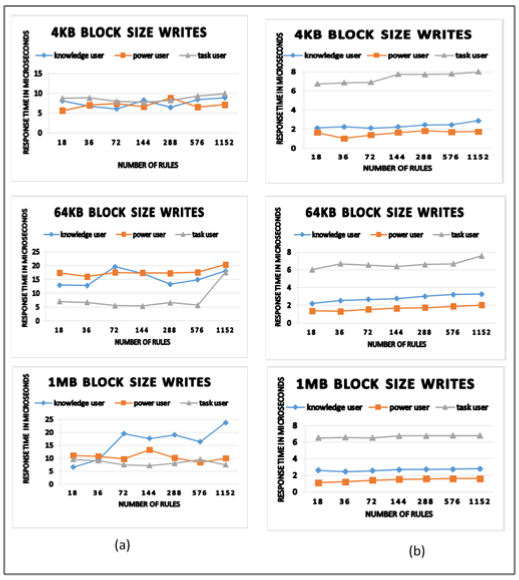


Fig.6. Response time comparison for reads (a) Without proposed solution and (b) With the proposed solution.

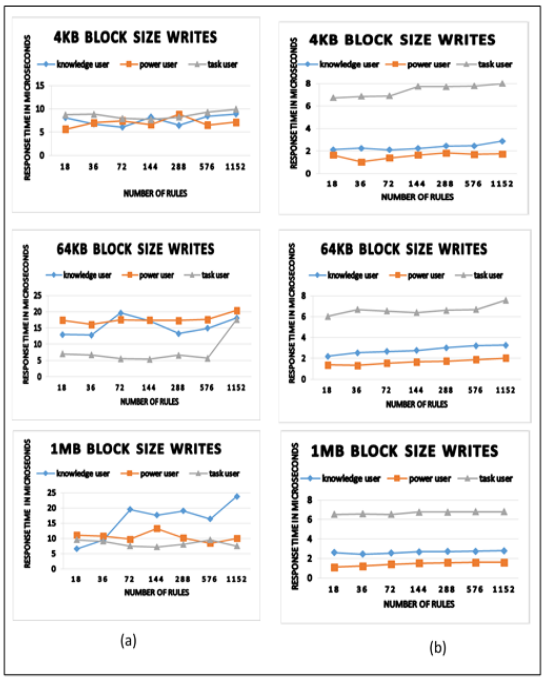


Fig.7. Response time comparison for writes (a) Without proposed solution and (b) With the proposed solution

From Figs.6(a) and 7(a) we further observe that the emulated classes of users are not able achieve their SLO with the list based classifier. However with the proposed solution as illustrated in Figs.6(b) and 7(b) all the classes of users are able to achieve an SLO close to the system being emulated[57]. These results are consistent with those in [9] where the authors varied the number of input output operations. In the experiments without the proposed solution, the latency was more compared to that without[57].

4.4. Classifier Accuracy

To evaluate for accuracy a file of 4GB was simulated for reads and writes with a queue depth of 32 and a block size of 64KB.The decision to use a block size of 64kb due to the fact that it's the default block size for windows. For queue depth of 32, it's the default in Parkdale and 4GB is the maximum file size possible for simulation while using Parkdale. After the reads and writes were completed the command *tc-s qdisc ls dev etho* was run on the router to generate the total packets generated and the classification per class[58]. Table 3 summarizes the results.

Table.3. Statistics of packet classification

	List Based U32 Classifier	Proposed Optimized Solution
Class	Number of Packets	Number of Packets
Power user	4871229	4973264
Knowledge user	4813052	4935052
Task user	4812035	4922035
Total number of packets classified	14496316	14830351
Total number of packets classified	8117936	13199012

With the list based classifier, the accuracy is 56 percent as compared to that of optimized solution of 89%. There is an improvement of 33% in the classification accuracy.

5. Conclusion and Future Scope

This paper embarked on the problem of performance isolation. Performance isolation is the segregation of traffic in order to prevent interference between classes traffic and also offer differentiated services. To achieve performance isolation packets form initiators needs to be classified for them to be offered differentiated treatment. During classification arriving packets are sequentially compared against a list of rules until a match is found. Due to increase in network speeds and mix of traffic in IP SANs it’s important for packet classifiers to inspect packets as fast as possible.

The paper has discussed in details the problems associated with linear search and techniques for optimizing linear search. The paper presents a solution for optimizing packet classification process for purposes of achieving performance isolation through throttling of flows.

The proposed solution has been tested and compared with traditional implementation of a linear search based classifier and established that the proposed solution gives better performance in terms of throughput and response time when used to classify traffic for performance isolation. The proposed solution was tested on an IPSAN and was found to be more suitable than the traditional implementation of linear search.

In future we would also like to explore techniques of using performance isolation in providing availability and reliability guarantees. In addition, we would like to implement our proposed solution in non-storage systems where performance isolation is required.

References

[1] Barzegaran, Mohammadreza, Anton Cervin, and Paul Pop. "Performance optimization of control applications on fog computing platforms using scheduling and isolation." IEEE Access 8 (2020): 104085-104098.

[2] J. Danielsson, T. Seceleanu, M. Jagemar, M. Behnam, and M. Sjodin, “Testing Performance-Isolation in Multi-Core Systems,” *2019 IEEE 43rd Annu. Comput. Softw. Appl. Conf.*, vol. 1, pp. 604–609, 2019.

[3] Riggio, Roberto, Francesco De Pellegrini, and Domenico Siracusa. "The price of virtualization: Performance isolation in multi-tenants networks." In 2014 IEEE Network Operations and Management Symposium (NOMS), pp. 1-7. IEEE, 2014.

[4] Muhammad Usman Ashraf, Sabah Arif, Abdul Basit, Malik Sheraaz Khan, "Provisioning Quality of Service for Multimedia Applications in Cloud Computing", International Journal of Information Technology and Computer Science, Vol.10, No.5, pp.40-47, 2018.

[5] Nam, Yoonsung, Yongjun Choi, Byeonghun Yoo, Hyeonsang Eom, and Yongseok Son. "EdgeIso: Effective Performance Isolation for Edge Devices." In 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 295-305. IEEE, 2020.

[6] A. Blenk, W. Kellerer, and S. Schmid, “On The Impact of the Network Hypervisor on Virtual Network Performance,” *2019 IFIP Netw. Conf. (IFIP Networking)*, pp. 1–9, 2019.

[7] Lumb, Christopher R., Arif Merchant, and Guillermo A. Alvarez. "Façade: Virtual Storage Devices with Performance Guarantees." In FAST, vol. 3, pp. 131-144. 2003.

[8] M. Karlsson, C. Karamanolis, and X. Zhu, “Triage: Performance Differentiation for Storage Systems Using Adaptive Control,” *ACM Trans. Storage*, vol. 1, no. 4, pp. 457–480, 2005.

[9] Gulati, Ajay, Irfan Ahmad, and Carl A. Waldspurger. "PARDA: Proportional Allocation of Resources for Distributed Storage Access." In FAST, vol. 9, pp. 85-98. 2009.

[10] Gulati, Ajay, Arif Merchant, and Peter J. Varman. "pClock: an arrival curve based approach for QoS guarantees in shared storage systems." ACM SIGMETRICS Performance Evaluation Review 35, no. 1 (2007): 13-24.

[11] Wachs, Matthew, Michael Abd-El-Malek, Eno Thereska, and Gregory R. Ganger. "Argon: Performance Insulation for Shared Storage Servers." In FAST, vol. 7, pp. 5-5. 2007.

[12] Kim, Hwajung, Heon Young Yeom, and Yongseok Son. "An i/o isolation scheme for key-value store on multiple solid-state drives." In 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W), pp. 170-175. IEEE, 2019.

[13] Gulder, Semra, and Mathieu Déziel. "Quality of Service Mechanism for MANET using Linux." In Proc. INSC Symposium,

NATO C3 Agency. 2003.

[14] Chang, Hsung-Pin, Yu-Cheng Yu, and Pei-Yao Chung. "Design and implementation of a shared multi-tiered storage system." In 2018 3rd International Conference on Computer and Communication Systems (ICCCS), pp. 94-98. IEEE, 2018.

[15] Almesberger, Werner. "Linux Network Tra c Control| Implementation Overview." Available at EPFL ICA http://diffserv. sourceforge. net (1999).

[16] Madhumita Kathuria, Sapna Gambhir,"Performance Optimization in WBAN Using Hybrid BDT and SVM Classifier", International Journal of Information Technology and Computer Science, Vol.8, No.12, pp.83-90, 2016.

[17] Hamed, Hazem, and Ehab Al-Shaer. "Dynamic rule-ordering optimization for high-speed firewall filtering." In Proceedings of the 2006 ACM Symposium on Information, computer and communications security, pp. 332-342. 2006.

[18] A. El-Atawy, T. Samak, E. Al-Shaer, and L. Hong, "Using online traffic statistical matching for optimizing packet filtering performance," *Proc. - IEEE INFOCOM*, pp. 866–874, 2007.

[19] A. Ganesh, A. Sudarsan, A. K. Vasu, D. Ramalingam, and T. Nadu, "I MPROVING F IREWALL P ERFORMANCE BY USING A C ACHE," vol. 7, no. 5, pp. 1594–1607, 2014.

[20] C. Wang, D. Sun, Y. Chai, and F. Zhou, "Enabling Accurate Performance Isolation on Hybrid Storage Devices in Cloud Environment," pp. 2018–2021, 2018.

[21] Cherian, Mimi, and Madhumita Chatterjee. "Optimized Firewall with Traffic Awareness." Int. J. Comput. Networks Appl 3, no. 2 (2016): 32-37.

[22] Z. Trabelsi and S. Zeidan, "Multilevel early packet filtering technique based on traffic statistics and splay trees for firewall performance improvement," *IEEE Int. Conf. Commun.*, pp. 1074–1078, 2012.

[23] Suresh, Nagulavancha, and B. Mathura Bai. "Predictive Modelling of Tree Rule Firewall for the Efficient Packet Filtering." International Journal of Computer Science and Information Security 14, no. 10 (2016): 189.

[24] H. Named and E. Al-Shaer, "Dynamic rule-ordering optimization for high-speed firewall filtering," *Proc. 2006 ACM Symp. Information, Comput. Commun. Secur. ASIACCS '06*, vol. 2006, pp. 332–342, 2006.

[25] A. K. Vasu and A. Ganesh, "Improving Firewall Performance by Eliminating Redundancies In Access Control Lists," *Priya Ayyappan Anirudhan Sudarsan Int. J. Comput. Networks*, no. 6, p. 92, 2014.

[26] Acharya, Subrata, Bryan N. Mills, Mehmud Abliz, Taieb Znati, Jia Wang, Zihui Ge, and Albert G. Greenberg. "OPTWALL: A Hierarchical Traffic-Aware Firewall." In NDSS. 2007.

[27] M. Abbasi and A. Shokrollahi, "Enhancing the performance of decision tree-based packet classification algorithms using CPU cluster," *Cluster Comput.*, vol. 7, 2020.

[28] Mini, T. V., R. Nedunchezhian, and V. Vijayakumar. "Development of an efficient association rule classifier with temporal characteristics and hierarchical partitioning." In 2016 Eighth International Conference on Advanced Computing (ICoAC), pp. 19-25. IEEE, 2017.

[29] X. He, T. Chomsiri, P. Nanda, and Z. Tan, "Improving Cloud Network Security Using Tree-Rule Firewall," *Futur. Gener. Comput. Syst.*, 2013.

[30] S. Zhao and J. Li, "An Efficient Tuple Pruning Scheme for Packet Classification Using On-chip Filtering and Indexing," *NOMS 2018 - 2018 IEEE/IFIP Netw. Oper. Manag. Symp.*, pp. 1–7, 2018.

[31] W. Yu, S. Sivakumar, and D. Pao, "Pseudo-TCAM: SRAM-Based Architecture for Packet Classification in One Memory Access," *IEEE Netw. Lett.*, vol. 1, no. 2, pp. 89–92, 2019.

[32] X. He, T. Chomsiri, P. Nanda, and Z. Tan, "Improving cloud network security using the Tree-Rule firewall," *Futur. Gener. Comput. Syst.*, 2013.

[33] S. Yingchareonthawornchai, J. Daly, A. X. Liu, and E. Torng, "A Sorted-Partitioning Approach to Fast and Scalable Dynamic Packet Classification," *IEEE/ACM Trans. Netw.*, vol. PP, pp. 1–14, 2018.

[34] Shen, Tong, Da-Fang Zhang, Gao-Gang Xie, and Xin-Yi Zhang. "Optimizing multi-dimensional packet classification for multi-core systems." Journal of Computer Science and Technology 33, no. 5 (2018): 1056-1071.

[35] Acharya, Subrata, Jia Wang, Zihui Ge, Taieb Znati, and Albert Greenberg. "Simulation study of firewalls to aid improved performance." In 39th Annual Simulation Symposium (ANSS'06), pp. 8-pp. IEEE, 2006.

[36] Abbasi, Mahdi, and Milad Rafiee. "A calibrated asymptotic framework for analyzing packet classification algorithms on GPUs." The Journal of Supercomputing 75, no. 10 (2019): 6574-6611.

[37] Narodytska, Nina, Leonid Ryzhyk, Igor Ganichev, and Soner Sevinc. "BDD-Based Algorithms for Packet Classification." *In 2019 Formal Methods in Computer Aided Design (FMCAD)*, pp. 64-68. IEEE, 2019.

[38] Zhao, Liang, Yuji Inoue, and Hideo Yamamoto. "Delay Reduction for Liner-Search Based Packet Filters." In ITC-CSCC: *International Technical Conference on Circuits Systems, Computers and Communications*, pp. 160-163. 2004.

[39] Storage Performance Council", Spcresults.org, 2021. [Online]. Available: http://spcresults.org/. [Accessed: 17- Jul- 2021].

[40] Live Optics - Real-world data for IT decisions : Live Optics", Liveoptics.com, 2021. [Online]. Available: https://www.liveoptics.com/. [Accessed: 17- Jul- 2021].

[41] Kim, Youngjae, Sudhanva Gurumurthi, and Anand Sivasubramaniam. "Understanding the performance-temperature interactions in disk i/o of server workloads." In The Twelfth International Symposium on High-Performance Computer Architecture, 2006., pp. 176-186. IEEE, 2006.

[42] Indira, B., K. Valarmathi, and D. Devaraj. "An approach to enhance packet classification performance of software-defined network using deep learning." Soft Computing 23, no. 18 (2019): 8609-8619.

[43] S. Hung, N. Iliev, B. Vamanan, A. R. Trivedi, H. I. S. Elf, and R. M. Ap, "Self-Organizing Maps-based Flexible and High-Speed Packet Classification in Software Defined Networking," *2019 32nd Int. Conf. VLSI Des. 2019 18th Int. Conf. Embed. Syst.*, pp. 545–546, 2019.

[44] Li, Rui, Bohan Zhao, Ruixin Chen, and Jin Zhao. "Taming the Wildcards: Towards Dependency-free Rule Caching with FreeCache." In 2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS), pp. 1-10. IEEE, 2020.

[45] Li, Wenjun, Tong Yang, Ori Rottenstreich, Xianfeng Li, Gaogang Xie, Hui Li, Balajee Vamanan, Dagang Li, and Huiping Lin.

"Tuple space assisted packet classification with high performance on both search and update." *IEEE Journal on Selected Areas in Communications* 38, no. 7 (2020): 1555-1569.

[46] W. Li, T. Yang, Y. Chang, T. Li, and H. Li, "TabTree : A TSS-assisted Bit-selecting Tree Scheme for Packet Classification with Balanced Rule Mapping," *2019 ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, pp. 1–8, 2019.

[47] W. Li, X. Li, H. Li, and G. Xie, "CutSplit : A Decision-Tree Combining Cutting and Splitting for Scalable Packet Classification," *IEEE INFOCOM 2018 - IEEE Conf. Comput. Commun.*, pp. 2645–2653, 2018.

[48] Li, Wenjun, Dagang Li, Yongjie Bai, Wenxia Le, and Hui Li. "Memory-efficient recursive scheme for multi-field packet classification." IET Communications 13, no. 9 (2019): 1319-1325.

[49] Li, Chuanhong, Xuewen Zeng, Lei Song, and Yan Jiang. "A Fast, Smart Packet Classification Algorithm Based on Decomposition." Journal of Control Science and Engineering 2020 (2020).

[50] Zhao, Liang, Akira Shimae, and Hiroshi Nagamochi. "Linear-tree rule structure for firewall optimization." In Communications, Internet, and Information Technology, pp. 67-72. 2007.

[51] T. Harada, K. Tanaka, and K. Mikawa, "Acceleration of Packet Classification via Inclusive," *2018 IEEE Conf. Commun. Netw. Secur.*, no. 3, pp. 1–2, 2018.

[52] Li, Chenglong, Tao Li, Junnan Li, Zilin Shi, and Baosheng Wang. "Update Latency Optimization of Packet Classification for SDN Switch on FPGA." In 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 213-213. IEEE, 2020.

[53] Li, Chenglong, Tao Li, Junnan Li, Zilin Shi, and Baosheng Wang. "Enabling Packet Classification with Low Update Latency for SDN Switch on FPGA." Sustainability 12, no. 8 (2020): 3068.

[54] Li, Chenglong, Tao Li, Junnan Li, Dagang Li, Hui Yang, and Baosheng Wang. "Memory optimization for bit-vector-based packet classification on FPGA." Electronics 8, no. 10 (2019): 1159.

[55] Kekely, Michal, Lukáš Kekely, and Jan Kořenek. "General memory efficient packet matching FPGA architecture for future high-speed networks." Microprocessors and Microsystems 73 (2020): 102950.

[56] B. Rouzbehani, L. M. Correia, and L. Caeiro, "An Optimised RRM Approach with Multi-Tenant Performance Isolation in Virtual RANs," *IEEE Int. Symp. Pers. Indoor Mob. Radio Commun. PIMRC*, vol. 2018-Septe, pp. 6–11, 2018.

[57] Q. Pan, K. Huang, H. Tang, and W. You, "A network slicing deployment method for guaranteeing service performance," *2018 IEEE 4th Int. Conf. Comput. Commun. ICCC 2018*, pp. 579–584, 2018.

[58] H. U. A. Wang, W. Rafique, and Z. Anwar, "Cyberpulse : A Machine Learning Based Link Flooding Attack Mitigation System for Software Defined Networks," *IEEE Access*, vol. 7, pp. 34885–34899, 2019.

Authors' Profiles

Joseph Kithinji is a Computer Science PhD candidate, School of Computing and Informatics at Meru University, Kenya. He holds an MSc degree in Data communication from KCA University and a BSc (honors) degree in Computer Science & Mathematics from Maseno University. He has ten years of teaching experience.

Dr. Makau S. Mutua holds PhD in Systems Analysis and Integration, a Master in Information Technology and Bachelor of Science in Computer Science. He is a Senior lecturer in Meru University of Science and Technology and currently serving as the dean of the School of Computing and Informatics. He is an established scholar and academician with several publications in refereed journals and book chapters. His research interests are Neural Networks, Computer Networks and Data Science.

Gitonga D. Mwathi holds a Ph.D in Computer Science from University of Nairobi, Master's Degree in IT (Systems Security) from Strathmore University and a Bachelor's degree in Computer Science and Engineering from Maseno University.He is currently a Lecturer at Chuka University with a teaching experience of twelve years at University level. His major research interests are in Computer Networks, Distributed Systems and Systems Security.

How to cite this paper: Joseph Kithinji, Makau S. Mutua, Gitonga D. Mwathi, "An Enhanced List Based Packet Classifier for Performance Isolation in Internet Protocol Storage Area Networks", International Journal of Information Technology and Computer Science(IJITCS), Vol.13, No.5, pp.51-63, 2021. DOI: 10.5815/ijitcs.2021.05.05