International Journal of Computer Applications (0975 – 8887)
Volume 76– No.4, August 2013

A Novel Technique for Fast Parallel Packet Classification

Balasaheb S. Agarkar
Department of Electronics and Telecommunication Engineering, SRES's College of Engineering, Kopargaon 423603, University of Pune, Pune, India

Uday V. Kulkarni, Ph.D
Department of Computer Science and Engineering, SGGS Institute of Engineering and Technology, Nanded 431 606, India

ABSTRACT

Packet classification is one of the most important enabling technologies for next generation network services. Four of the main challenges in the packet classification are increase in size of the classifier, link speed, amount of multimedia traffic, and number of media-rich and bandwidth intensive internet applications. Due to this there is a need of memory efficient and high throughput packet classification schemes.

In this paper a novel technique for fast parallel packet classification (FPPC) is proposed. A recent paper [14] showed how to construct a hierarchical $Tree - Trie_{\epsilon}$ (TT_{ϵ}) search structure and a clustering algorithm that partitions a given classifier into a fixed number of clusters. This dramatically enhances memory efficiency and throughput. This idea is extended to address the more challenging problem of general packet classification. The hierarchical search results are passed on to the bloom filter for final classification. Also it is observed that in a large classifier many rules have very poor hit rate. If Top-N selection approach is used, without affecting minimum Quality of Service (QoS) requirements it is possible to reduce mean delay and increase the throughput. The simulation results shows that proposed scheme gives 22.5% rise in the throughput and 5.62% decrease in mean delay with slight decrease in memory efficiency as compared to Hierarchical Hybrid Search Structure (HHSS) scheme.

General Terms

Computer Networks, Internet Traffic

Keywords

Bloom filter, Classifier, Hierarchical structure, Packet classification.

1. INTRODUCTION

The process of mapping packets to different service classes or flows in an internet router is referred to as packet classification. Flow is defined as the collection of all the packets which obey a same predefined rule. Traditional routers do not provide service differentiation because they treat all traffic going to a particular Internet address in the same way. Packet classification is an enabling function for a variety of Internet applications including QoS, security, traffic monitoring, Virtual Private Networks (VPN) and multimedia communications. In order to classify a packet as belonging to a particular flow routers must perform a search over a set of filters or rules also known as classifier using multiple fields of the packet header as the search key. The multi-field or multi-dimensional packet classification becomes a critical operation in networking devices such as routers. In general, there have been two major threads of research addressing packet classification: algorithmic and architectural [1]. A few pioneering groups of researchers posed the problem, provided complexity bounds, and offered a collection of algorithmic solutions [2], [3]. Subsequently, the design space has been vigorously explored by many offering new algorithms and improvements upon existing algorithms [4-7]. Given the inability of early algorithms to meet performance constraints imposed by high speed links, researchers in industry and academia devised architectural solutions to the problem. This thread of research produced the most widely used packet classification device technology, Ternary Content Addressable Memory (TCAM) [8], [9]. New architectural research in this area combines intelligent algorithms and novel architectures to eliminate many of the unfavorable characteristics of current TCAMs. It is observed that the community appears to be converging on a combined algorithmic and architectural approach [10], [11].

The growing complexity of the Internet is creating new applications, placing additional demands on the packet classification subsystem of routers and other packet handling devices. Several protocols and techniques such as DiffServ or NSLP (Network Address Translation (NAT) / Firewall NSIS Signaling protocol) assumes that the information relevant to packet classification is contained in five or less number of IPv4-fields namely source IP address (32 bits), destination IP address (32 bits), source port number (16 bits), destination port number (16 bits), and the protocol (8 bits).

1. 1 Need of Packet Classification

Packet classification is very important in improving the performance of internet traffic.

Technology innovations of network systems: Network Processor Unit (NPU) has emerged as a promising candidate for a networking system building block. NPU opens a new venture to explore advanced technologies such as multi-core network processors, thread-level parallelism to attack the performance bottleneck of classification. Also they provide us with unprecedented computing power, as well as highly integrated resources. Hence novel packet classification solutions must be well suited for the advanced hardware and software technologies to break the bottleneck, providing the availability of these innovative technologies to a vast majority of customers. NPU has potential to provide total solution for packet processing, including forwarding and classification [12], [13].

Ever-increasing complexity of network applications: The growth and diversification of the Internet imposes increasing demands on the performance and functionality of network infrastructure. Traditional packet classification is mainly employed by firewalls to screen unwanted traffics. With more and more network applications implemented in today's networking devices, packet classification is widely used for various kinds of applications, such as service-aware routing, intrusion prevention and traffic shaping. Thus, novel solutions must be more intelligent to effectively handle multifarious types of rule sets without significant loss of performance [13].

1.2 Types of Packet Classification

Packet classification approaches can be classified into four main groups; exhaustive search, decomposition, decision tree, and hierarchical-trie (H-trie) [14].

For any searching problem the most rudimentary solution is simply to search through all the entries. In exhaustive search, all the entries in a rule set are analyzed [15]. The two basic approaches in this group are linear search and TCAM based search. Linear search is performed by comparing the header of a packet with all the entries in a rule set sequentially. Linear search becomes a slow process for large rule sets. Hence it is popular for the final stage of a search when the set of possible matching rules has been reduced to a bounded constant. TCAM devices allow a parallel search over all rules in the classifiers. In this the header of a packet is compared with all the entries in parallel. TCAMs have several disadvantages such as high cost, storage inefficiency, high power consumption, and limited scalability to long input keys. The storage inefficiency occurs due to the need of conversion of ranges into prefixes.

In decomposition based solutions, independent searches on each header field are performed, and then the results are combined. These approaches offer high throughput but require high amount of storage space in order to aggregate the results of single searches efficiently. The primary challenge for these approaches is how to efficiently aggregate the results of the single field searches.

Most of the proposed packet classification algorithms and architectures are based on decision trees, which take the geometric view of the packet classification problem. HiCuts and its enhanced version HyperCuts are representatives of such algorithms. At each node of the decision tree, the search space is cut based on the information from one or more fields in the rule. HiCuts builds a decision tree using local optimization decisions at each node to choose the next dimension to test, and how many cuts to make in the chosen dimension. The HyperCuts algorithm allows cutting on multiple fields per step, resulting in a fatter and shorter decision tree. The common problem of these approaches is that it is difficult to support incremental updates.

Hierarchical-trie (H-trie) is built using the source IP address (S_{ad}) and destination IP address (D_{ad}) prefixes. Initially, a S_{ad} trie is constructed using all the S_{ad} prefixes. For each prefix node in S_{ad} trie, a D_{ad} trie is constructed using D_{ad} prefix(es) associated with that S_{ad} prefix. Thus, the structure consists of a large S_{ad} trie and hierarchically connected multiple small D_{ad} tries. Search starts from the S_{ad} trie. If a prefix node of the S_{ad} trie is visited, then the corresponding D_{ad} trie connected to that prefix node is traversed. Even though a match can be found at any node in the D_{ad} trie, search has to backtrack to the S_{ad} trie and continue the search to find other possible matches. The search terminates after each leaf node in the S_{ad} trie is visited. Set-pruning trie eliminates the backtracking by replicating the rules. Grid-of-Tries (GoT) data structure for 2-field packet classification eliminates the backtracking by introducing switch pointers to some trie nodes and hence each rule is stored at only one node. Despite of good memory efficiency, extension of the GoT to multiple fields is not clear.

1.3 Issues of Packet Classification

Continual growth of network bandwidth is the very important issue of packet classification. The explosion in demand for network bandwidth owes much to the growth in data traffic. Leading service providers report bandwidths doubling on their backbones about every six to nine months [13].

Power consumption is also another important issue of packet classification. As routers achieve aggregate throughputs of trillions of bits per second, power consumption becomes an increasingly critical concern. Both the power consumed by the router itself and the infrastructure to dissipate the tremendous heat generated by the router components significantly contribute to the operating costs. Given that each port of high-performance routers must contain route lookup and packet classification devices, the power consumed by search engines is becoming an increasingly important evaluation parameter [1], [15].

Speed and flexibility in specifications is also another issue in packet classification. In the general packet classification problem, packets are classified according to a set of packet filters, which define patterns that are matched against incoming packets. Typically, packet filters specify possible values of the source and destination address fields of the IP header, the protocol field (often including flags) and the source and destination port numbers (for TCP and UDP). The address fields are often specified as address prefixes, although arbitrary bit masks of the address fields are commonly allowed in packet filters and this feature is used in real filter sets, although relatively infrequently. Filters typically specify a range of port numbers for matching packets. Protocols can be either specified exactly or as a wildcard. Some systems allow protocol values to be specified by bit masks as well, although it’s not clear how useful that feature is [1], [15].

The following sections are organized as follows: section 2 gives literature review of packet classification; section 3 identifies problem; the proposed algorithm FPPC is described in section 4 and evaluated in section 5 and concluded in section 6.

2. LITERATURE REVIEW

Yaxuan Qi et al., [13] have proposed a novel packet classification algorithm named HyperSplit. Compared to the well-known HiCuts and HSM algorithms, HyperSplit achieves superior performance in terms of classification speed, memory usage and preprocessing time. In this paper, they combined the advantages of existing algorithms: rule-based space decomposition and local-optimized recursion. This algorithm guarantees explicit worst-case classification speed and explores the data redundancy in rule sets to reduce memory usage. The data structure of HyperSplit is also carefully designed for efficient storage and fast access. The practicability of the proposed algorithm is manifested by two facts in their test: HyperSplit is the only algorithm that can successfully handle all the rule sets; HyperSplit is also the only algorithm that reaches more than 6 Gbps throughput on the Octeon3860 multi-core platform when tested with 64-byte Ethernet packets against 10K ACL rules.

Oguzhan Erdem et al., [14] have proposed a high-throughput and memory-efficient SRAM-based linear pipelined architecture for packet classification. A clustering algorithm that partitions a given rule database into a fixed number of clusters to eliminate backtracking in the state-of-the-art hierarchical search structure. A special type of ternary trie data structure (T_{ϵ}) and a two-stage hierarchical search structure that achieves substantial memory saving in hardware implementation are used. Their design achieves memory efficiency between 10.37 and 22.81 bytes of memory per rule, and sustains a high throughput of 418 million packets per second on a state-of-the-art FPGA device.

Weirong Jiang et al., [15] have proposed Decision-tree-based, two-dimensional dual-pipeline architecture for multi-field packet classification. To fit the current largest rule set in the on-chip memory of the FPGA device, they propose several optimization techniques for the state of-the-art decision-tree-based algorithm, so that the memory requirement is almost linear with the number of rules. Specialized logic is developed to support varying number of branches at each decision tree node. A tree-to-pipeline mapping scheme is carefully designed to maximize the memory utilization. Since their architecture is linear and memory based, on-the-fly update without disturbing the ongoing operations is feasible.

Yadi Ma and Suman Banerjee, [16] have proposed a practical and efficient solution which introduces a smart pre-classifier to reduce power consumption of TCAMs for multidimensional packet classification. They reduce the dimension of the problem through the pre-classifier which pre-classifies a packet on two header fields, source and destination IP addresses. Then they return to the higher dimension problem where only a small portion of a TCAM is activated and searched for a given packet. The smart pre-classifier is built in a way such that a given packet matches at most one entry in the pre-classifier, which make commodity TCAMs sufficient to implement the pre-classifier. Furthermore, each rule is stored only once in one of the TCAM blocks, which avoids rule replication. The presented solution uses commodity TCAMs and the proposed algorithms are easy to implement. Their scheme achieves a median power reduction of 91% and an average power reduction of 88% on real and synthetic classifiers respectively.

Alan Kennedy et al., [17] have proposed low power architecture for a high speed packet classifier which could be used as an on-chip hardware accelerator for a network processor or as an external chip. This architecture uses an adaptive clocking unit to exploit the fluctuation in Internet traffic by reducing the clock frequency during times of low traffic and increasing the clock frequency at times of high traffic. In order to make the decision of frequency scaling, the fields of a packet header used for classification are extracted into a buffer upon its arrival and the queue length is monitored. The hardware accelerator implements a modified version of the HiCuts and HyperCuts packet classification algorithms. Instead of comparing a large number of rules simultaneously (as is the case with TCAM), the algorithms divide the hyperspace of the rule set heuristically into multiple groups so that each subset contains only a small number of rules that are suitable for linear search, reducing the unnecessary comparisons and thus the power consumption. The hardware accelerator utilizes the flexibility of a FPGA's block RAM by using SRAM with long word line to reduce the number of clock cycles needed to perform a linear search on the selected rules.

3. PROBLEM IDENTIFICATION

The approach which is proposed in [14] is based on Hierarchical search structure. In this one Hierarchical TT_ϵ structure is used for packet classification. This approach eliminates the need for backtracking in hardware implementation and achieves substantial memory saving. The following drawbacks are observed in this approach:

- Low throughput with algorithmic implementation.
- Mean delay per packet from source to destination is more.
- Potentially long latency for rule sets with large number of overlapped rules.
- The memory efficiency of the design can be negatively affected by new rule updates.
- The power consumption is more.

In order to solve some of the above problems, in this paper it is proposed to use a novel technique of fast parallel packet classification.

4. FAST PARALLEL PACKET CLASSIFICATION

4.1 Overview

In this paper, a novel technique for fast parallel packet classification for internet traffic is proposed. This technique involves the construction of Hierarchical tree architecture. The hierarchical search results are transmitted to the bloom filter for performing packet classification. In this hash values are computed which sets the bits into bit array during construction of data structure. The bits in the bit array are analyzed during classification and upon observing the first unset bit, the analysis is stopped. This process minimizes the power consumed also during hashing function. Bloom filter has the drawback of false positives but its probability can be minimized by proper design of bloom filter.

4.2 Top-N Selection Algorithm

This technique involves the selection of a subset of N rules with maximum hit-rates as per the traffic scenario. The hit rate represents the fraction of packet classification queries which is acquired using hit-counts situated in the majority switches and also it is a component of open flow specification.

By sorting the hit rate table in descending order of hit-rates, the first N rules are the target rules and together are called as the target set. Since some of the target rules may depend on rules that are not in the target set, one cannot simply select the N target rules as the top-N rules [18].
As far as the selection strategy is concerned each target rule is examined in order to decide either to include in the top-N list and what position to place it. Let R is original rule set, G is dependency graph and T is derived rule set. Then the detailed procedure for Top-N Selection can be given as follows in algorithm 1.

Algorithm 1: Top-N Selection

```
1. Procedure Top-$N$ Selection
2. Select $N$ target rules with highest reference
3. Sort them to descending priority
4. for each target rule $T_i$ do
5. $T_i = partition\_decision\ (R_i , G)$
6. if Top-$N$ list is full then
7. Stop and output the Top-$N$ list
8. else
9. put $T_i$ into Top-$N$ list
```

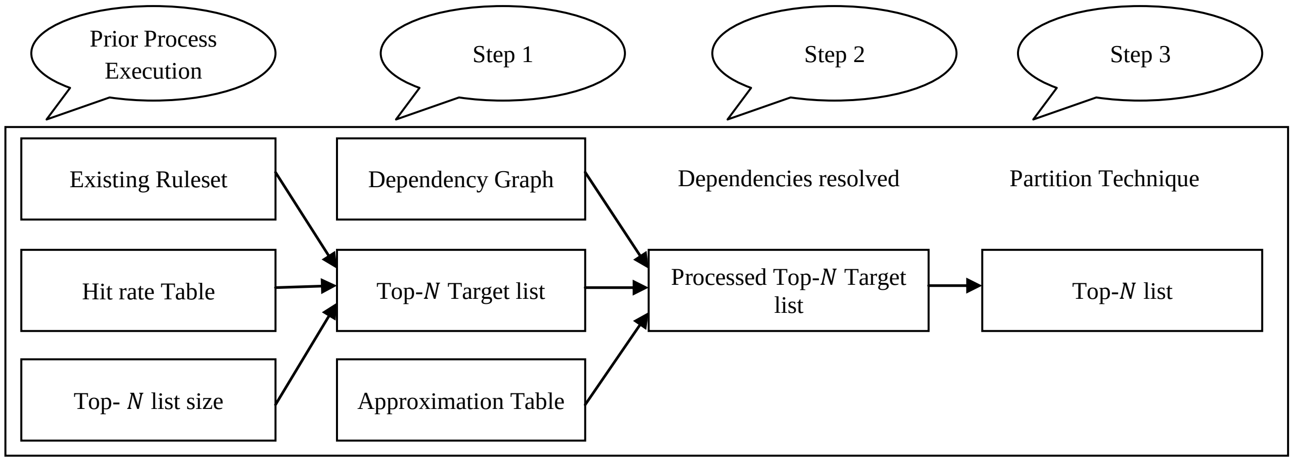

Fig 1: Flow diagram of Top-N selection algorithm

Conflicts can be resolved by splitting the target rule concerned into smaller derived rules that are disjoint with the dependent rules. It is noteworthy that in either way, some target rules, starting from bottom of the target list, have to be excluded from the sub-rule set because only N rules are permitted in the sub-rule set. This unavoidably lowers the overall hit-rate provided by the resulting sub-rule set. The proper
choice between the two options mainly depends on; The number of derived rules that are required to resolve the dependency, the total hit-rate offered by the dependent rule(s); and the total hit-rate of the target rules that would be excluded in each options.
The working of Top-N selection algorithm can be clearly understood from fig 1.

4.3 Proposed Technique

Let S_{ad} and D_{ad} be the source and destination IP addresses respectively, S_{pn} and D_{pn} represent the port numbers of source and destination respectively, P_{tcl} be the protocol, PT be the priority of the rule in the example classifier, HC be the hit count, S be the set of S_{ad} prefixes, P_s be the number of prefixes in S, D be the set of D_{ad} prefixes, P_d be the number of prefixes in D, C be the number of clusters, S_i be the sub set of S in cluster i ($1 \le i \le C$), R_{trie} be the upper bound for the number of rules per trie node.

Table 1. An example classifier

Rule	S_{ad}	D_{ad}	S_{pn}	D_{pn}	P_{tcl}	PT	HC	Action
R_1	00*	00*	*	80	TCP	1	2	Act0
R_2	0*	0*	17	*	UDP	2	3	Act1
R_3	10*	10*	*	*	TCP	2	1	Act2
R_4	11*	10*	*	100	TCP	3	3	Act3
R_5	11*	1*	*	*	*	4	1	Act4
R_6	*	11*	17	44	UDP	5	0	Act5
R_7	0*	10*	80	*	TCP	6	6	Act6
R_8	0*	01*	17	17	UDP	6	7	Act7
R_9	0*	1*	44	*	TCP	7	5	Act8
R_{10}	00*	1*	17	44	UDP	7	5	Act9
R_{11}	00*	11*	*	100	TCP	8	6	Act10
R_{12}	10*	1*	*	*	*	9	7	Act11
R_{13}	*	00*	*	*	TCP	7	5	Act12
R_{14}	0*	10*	*	100	TCP	5	5	Act13
R_{15}	0*	1*	*	*	TCP	0	6	Act14
R_{16}	0*	10*	17	17	UDP	4	7	Act15
R_{17}	111*	000*	80	*	TCP	6	7	Act16

Let SV be the skip value used in path compression, BS be the bit string used to store missing bits in path compression, ϵ_b be the upper bound for the number of consecutive ϵ transitions, α be the ratio of the number of rules stored in secondary memory over the total number of rules R in the given rule set, α_T be the upper bound for α. The sample rule set is given in table 1.

The pseudo-code for clustering is given in algorithms 2.

```
Algorithm 2: Clustering algorithm

Input: Prefix set $S$, Number of clusters $C$
Output: A partition of $S$ into a collection of non-empty
        prefix subsets $(S_i)$ such that within each subset
        all the prefixes are pair wise disjoint.
1.   $i = 1$
2.   Construct a binary trie using prefix set $S$
3.   while $i \le C$ do
4.   Move the leaves of the trie into $S_i$
5.   Trim the leaf-removed trie
6.   $i = i + 1$
7.   end while
8.   Leaf-push the trie and move the
     leaf-pushed leaves into $S_i$
9.   return $\{S_i\}$, $1 \le i \le C$
```

After this a hierarchical search structure is constructed which consists of two stages. In stage one a binary search tree is built for each cluster using S_{ad} prefixes. In the binary search tree each node consists of a value (prefix), a prefix length, left pointer, and right pointer. In stage two each node of S_{ad} tree connects to a D_{ad} trie. Hence in each cluster, the number of D_{ad} tries is equal to the number of S_{ad} prefixes. Each prefix node of a D_{ad} trie stores at least one rule. For each rule only the
S_{pn} , D_{pn} , P_{tcl} , and PT fields are stored. Here the over-lapped rules are stored in the D_{ad} trie nodes rather than pointing to a list of these rules as did by F. Baboescu, S. Singh, and G. Varghese in their paper Packet Classification for Core Routers: Is there an alternative to CAMs? [4].therefore, the matching results can be resolved at each node. However, the number of rules stored at each node is not constant. This leads to memory inefficiency for hardware implementation. To improve memory inefficiency, a special type of ternary trie data structure T_ϵ is used.

A single node in T_ϵ trie structure may have a single ϵ branch for which no input bit is consumed or '0' and/or '1' branch same as binary trie, but cannot have both at the same time. The main goal of utilizing the ϵ transition is to split a super-node in a trie into multiple small and fixed size nodes. These nodes are sequentially connected by the ϵ branches. In this data structure a limit is set on the number of overlapped rules per node, denoted by R_{trie} . The pseudo-code to construct T_ϵ structure is given in algorithm 3.

Algorithm 3: T_ϵ construction algorithm

```
Input: Prefix table consisting of prefixes $\{D_i\}, 0 \leq i < P_d$ with associated next hop info $NHI_i$
Input: Root node $D_{root}$ of $T_\epsilon$, $D_{root}.left = NULL$, $D_{root}.right = NULL$, $D_{root}.size = 0$
Input: Maximum node size, $R_{trie}$
Output: $T_\epsilon$ trie structure
1.   $i = 0$
2.   while $i \leq P_d$ do
3.   Binary_trie_insert ($D_i$, $D_{root}$)
4.   Let $D_{node}$ be a node where $D_i$ is stored
5.   if $D_{node}.size < R_{trie}$ then
6.   Store $NHI_i$ to $D_{node}$ , $D_{node}.size = D_{node}.size + 1$
7.   else
8.   if $D_{node}$ has $\epsilon$ branch ($D_{root}.left = \epsilon$ branch, $D_{node}.right = 0$) then
9.   $D_{node} = D_{node}.left$
10.  Go to step 5
11.  else
12.  Create a node $D_{new}$
13.  $D_{new}.left = D_{node}.left$
14.  $D_{new}.right = D_{node}.right$
15.  $D_{node}.left = D_{new}$
16.  $D_{node}.right = 0$
17.  Store $NHI_i$ to $D_{new}$, $D_{new}.size = D_{new}.size + 1$
18.  end if
19.  end if
20.  $i = i + 1$
21.  end while
22.  return $T_\epsilon$ trie
```

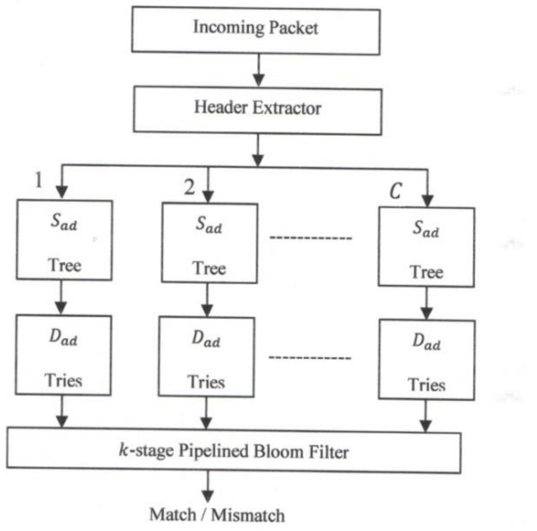

Fig 2: Hierarchical TT_ϵ search structure

The entire process is illustrated using the following architecture shown in fig 2. The incoming packet is passed on to the header extractor; from which the different field values are separated. These header field values are routed to all pipelines to perform the search. These results are transmitted to the Bloom filter for performing packet classification.

This process involves the following two phases. Initially, for a given set of rules R Bloom filter computes k hash values for each element r_i ranging from 1 to b using k hash functions, $h_1(), \ldots, h_k()$. Each of these values addresses a single bit in the b bit vector and sets it to one. Note that if one of the k hash values addresses a bit that is already set to 1, that bit is not changed. The following pseudo-code describes adding an element r to a Bloom filter.

```
BFAdd ($r$)
    1.  for ($i = 1\ to\ k$)
    2.  Vector [$h_i(r)$] ← 1
```

Querying the filter for set membership of a given element r is similar to the above process. Given element r, k hash values are generated using the same hash functions. The bits in the b-bit long vector at the locations corresponding to the k hash values are checked. If at least one of the k bits is 0, then the element is declared to be a nonmember of the set. If all the bits are found to be 1, then the element is said to belong to the set with a certain probability. If all the k bits are found to be 1 and x is not a member of R, then it is a false positive. The following pseudo-code describes the query process.

```
BFQuery ($x$)
    1.  for ($i = 1\ to\ k$)
    2.      if ($Vector[h_i(r)] = 0$) return false
    3.  return true
```

At the time of querying the filter in case of k-stage pipelined Bloom filter as soon as the first unset bit is found, the analysis is stopped. The same process is illustrated in the following sections.

4.3.1 k-Stage Pipelined Bloom Filter

The k-Stage Pipelined Bloom Filter is defined as the Bloom filter implementing its hash functions in a pipelined fashion [19]. The main advantage of this filter is that it prevents using subsequent stages for resolving the input as the member of bit-array. At worst, it operates like a standard Bloom filter that uses the entire hashing functions prior to deciding the input category.

Fig 3 shows the architecture of k-stage pipelined Bloom filter. It includes k set of hashing functions. The hash values are constantly computed in each stage for the given input. Also, only when the input matches with the bit-array sought, the subsequent stage computes hash values. The term En means enable. This reveals the matching criteria in the earlier stage enabling the next stage of the pipeline.

With regards to the programming stage, the entire hashing function is used and pipelined stages are permanently occupied. Hence k-stage pipelined bloom filter is used in membership checking stage which is shown below.

Let b represent the size of the bit-array and n be the input count. The probability that one of b bits is set using single hashing function working on a single input is given by equation 1.

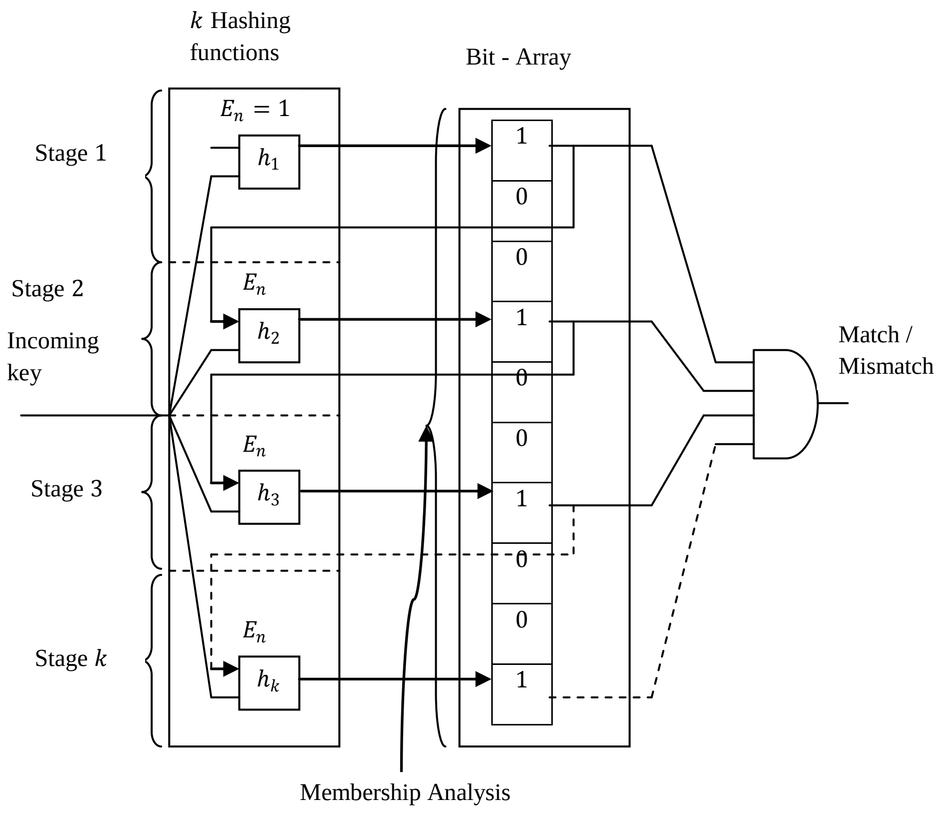

Fig 3: Architecture of k-stage pipelined bloom filter

$$p_s = \frac{1}{b} \qquad (1)$$

and the probability of unsetting a bit is computed as

$$p_{us} = \left[1 - \frac{1}{b}\right]. \qquad (2)$$

Similarly, subsequent to the process of programming, all the inputs into the pipelined bloom filter using k independent hashing functions, the probability of bit still remaining unset can be calculated using following equation.

$$p'_{us} = \left[1 - \frac{1}{b}\right]^{kn} \approx e^{\frac{-kn}{b}} \qquad (3)$$

Thus the probability by which the bit is set is

$$p'_s = [1 - p'_{us}] = 1 - e^{\frac{-kn}{b}}. \qquad (4)$$

4.3.1.1 Power Consumption

Let P_h be the power consumed by the hashing function, P_{lu} be the power consumed by the bit-array lookup function, $P_{\&k}$ be the power consumed by the k input AND operation.

The power consumed by pipelined bloom filter P_{pbf} is computed from fig 3 and is given by the following equation

$$\begin{aligned} P_{pbf} &= P_{hh_1} + P_{luh_1} + p'_{sh_1}\left(P_{hh_2} + P_{luh_2}\right) \\ &\quad + p'_{sh_1} p'_{sh_2}\left(P_{hh_3} + P_{luh_3}\right) + \cdots \\ &\quad + p'_{sh_1} \ldots\ldots p'_{sh_{k-1}}\left(P_{hh_k} + P_{luh_k}\right) + P_{\&k} \end{aligned}$$

$$\begin{aligned} &= P_{hh_1} + P_{luh_1} + B\left(P_{hh_2} + P_{luh_2}\right) + B^2\left(P_{hh_3} + P_{luh_3}\right) \\ &\quad + \cdots + B^{k-1}\left(P_{hh_k} + P_{luh_k}\right) + P_{\&k} \end{aligned}$$

$$= \sum_{i=1}^{k} B^{i-1}\left(P_{hh_i} + P_{luh_i}\right) + P_{\&k} \qquad (5)$$

Where $B = \left[1 - e^{\frac{-kn}{b}}\right]$.

The value p'_{sh_i} used in above equation (5) represents the probability of the bit which is set using hashing function with index i. Thus P_{pbf} is shown using the following equation

$$P_{pbf} = \sum_{i=1}^{k} \left[1 - e^{\frac{-kn}{b}}\right]^{i-1} \left(P_{hh_i} + P_{luh_i}\right) + P_{\&k}. \quad (6)$$

This approach of using k-stage pipelined bloom filter consumes minimum power.

4.4 Advantages

- The top N approximation algorithm keeps away rules having poor heat rate.
- The utilization of Bloom filters minimizes the processing time of packet classification.

- The overall power consumption is also minimized.

5. SIMULATION RESULTS

5.1 Simulation Model and Parameters

In this section, the performance of the

Fast Parallel Packet Classification (FPPC) algorithm is examined with an extensive simulation study by using NS-2 [20]. The results obtained are compared with HHSS. The topology used in the simulation is depicted in Fig 4. The UDP traffic flows are used. The filtersetsize is varied as 500, 1000, 1500, and 2000 number of rules. The filter sets are generated by using the ClassBench [21] tool.

5.2 Performance Metrics

In the simulation experiments, the following parameters for UDP CBR data flows are measured.

- Throughput (average no. of packets processed at the node per second)
- Delay (the overall mean delay occurred including searching and matching delays in µseconds)
- Efficiency (the memory utilized in terms of number of bits per rule)

The results are described in the next section.

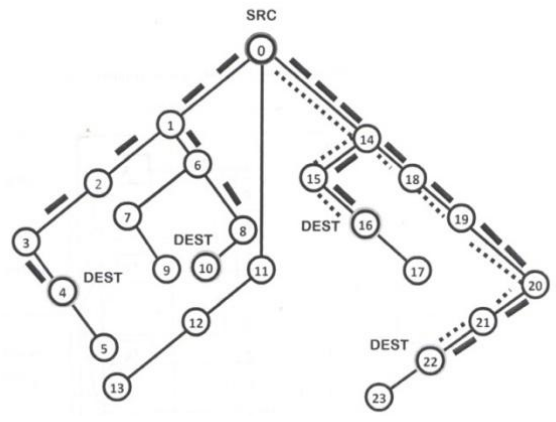

Fig 4: Simulation Topology

In this experimentation the database is generated by varying the filtersetsize. The size of the filerset is varied from 500 to 2000. In this case, a set of CBR flows using UDP protocol is transmitted from the same source to different destinations.

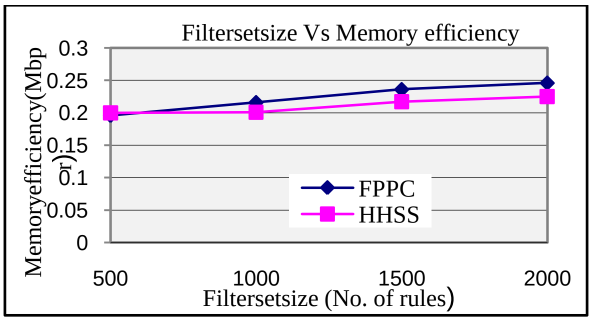

Fig 5: Filtersetsize Vs Efficiency

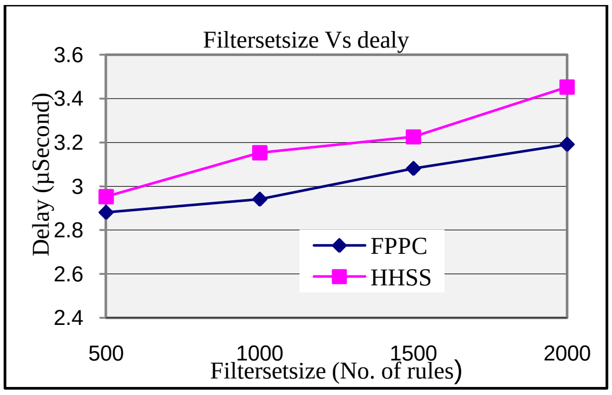

Fig 6: Filtersetsize Vs Delay

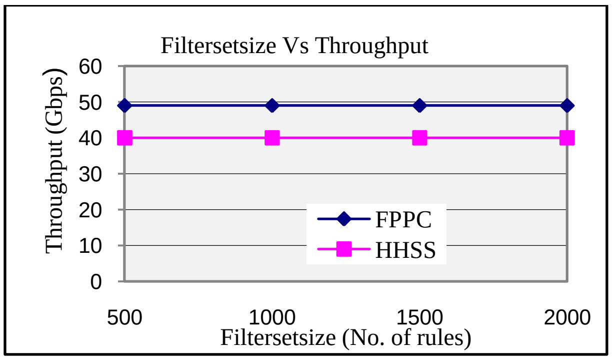

Fig 7: Filtersetsize Vs Throughput

When the filter size is increased, it results in the small degradation of efficiency. Fig 5 shows the results of efficiency when the filter size is increased. It is observed that the efficiency of FPPC is slightly less than that of HHSS, because of the increase in bit vector size in Bloom filter in FPPC classification.

The increase in filter size increases the size of database and hence the searching and matching delay tends to increase. It is observed from fig 6 that the delay of the proposed FPPC is less than the existing HHSS technique, since FPPC uses the Top-N rule selection approach.

The increase in filtersetsize does not make any significant variations in the overall throughput. This can be depicted from the fig 7. It is observed that FPPC has higher throughput as compared to HHSS.

6. CONCLUSIONS

In this paper, a design of novel technique for fast parallel packet classification for internet traffic is proposed. In this algorithm a hierarchical search structure constructed by using ternary trie data structure is used. This avoids backtracking which saves time. By simulation results, it can be concluded that the proposed technique minimizes the searching delay thereby increasing throughput with little decrease in memory efficiency. One of the drawbacks of the design is the potentially long latency for rule sets with large number of overlapped rules. In future it is planned to explore the methods to reduce latency of overlapped rules and update the overall performance of above technique.

7. REFERENCES

[1] David E. Taylor, "Survey & Taxonomy of Packet Classification Techniques," Technical Report WUCSE-2004-24, Department of Computer Science and Engineering, Washington, May 2004.

[2] V. Srinivasan, S. Suri, G. Varghese, and M. Waldvogel, "Fast and Scalable Layer Four Switching," in ACM SIGCOMM, June 1998.

[3] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," in ACM SIGCOMM, pp. 135–146, 1999.

[4] F. Baboescu, S. Singh, and G. Varghese, "Packet Classification for Core Routers: Is there an alternative to CAMs?" in IEEE INFOCOM, 2003.

[5] P. Gupta and N. McKeown, "Packet Classification using Hierarchical Intelligent Cuttings," in *Hot* Interconnects VII, August 1999.

[6] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet Classification Using Multidimensional Cutting," in Proceedings of ACM SIGCOMM, August 2003. Karlsruhe, Germany.

[7] T. Y. C. Woo, "A Modular Approach to Packet Classification: Algorithms and Results," in IEEE INFOCOM, March 2000.

[8] R. K. Montoye, "Apparatus for Storing "Don't Care"in Content Addressable Memory Cell." United States Patent 5,319,590, June 1994. HaL Computer Systems, Inc.

[9] Young-Deok Kim, Hyun-Seok Ahn, Suhwan Kim, and Deog-Kyoon Jeong, "A High-Speed Range-Matching TCAM for Storage-Efficient Packet Classification," IEEE Transactions on Circuits and Systems-I: Regular Papers, vol. 56, no. 6, June 2009.

[10] D. E. Taylor and J. S. Turner, "Scalable Packet Classification using Distributed Crossproducting of Field Labels," Tech. Rep. WUCSE-2004-38, Department of Computer Science and Engineering, Washington.

[11] J. van Lunteren and T. Engbersen, "Fast and scalable packet classification," IEEE Journal on Selected Areas in Communications, vol. 21, pp. 560–571, May 2003.

[12] Duo Liu, Zheng Chen, Bei Hua, Nenghai Yu and Xinan Tang, "High-Performance Packet Classification Algorithm for Multithreaded IXP Network Processor," ACM Transactions on Embedded Computing Systems, Vol. 7, no. 2, Article 16, February 2008.

[13] Yaxuan Qi, Lianghong Xu, Baohua Yang, Yibo Xue and Jun Li, "Packet Classification Algorithms: From Theory to Practice," IEEE INFOCOM 2009.

[14] O˜guzhan Erdem, Hoang Le and Viktor K. Prasanna, "Hierarchical Hybrid Search Structure for High Performance Packet Classification," in IEEE INFOCOM, 2012.

[15] Weirong Jiang and Viktor K. Prasanna, "Large-Scale Wire-Speed Packet Classification on FPGAs," in ACM, 2009.

[16] Yadi Ma and Suman Banerjee, "A Smart Pre-Classifier to Reduce Power Consumption of TCAMs for Multi-dimensional Packet Classification," in ACM, 2012.

[17] Alan Kennedy, Xiaojun Wang, Zhen Liu and Bin Liu, "Low Power Architecture for High Speed Packet Classification," in ACM, 2008.

[18] Ho-Yu Lam, Donghan Wang and H. Jonathan Chao, "A Traffic-aware Top-N Firewall Approximation Algorithm," in the first International workshop on security in computers, networking and communications, 2011.

[19] Mahmood Ahmadi and Stephan Wong, "K-Stage Pipelined Bloom Filter for Packet Classification," in International conference on computational science and engineering, 2009.

[20] Network simulator, http://www.isi.edu/nsnam/ns

[21] D. E. Taylor and J. S. Turner. "ClassBench: A Packet Classification Benchmark," IEEE/ACM Transactions on Networking, vol. 15, no. 3, pp. 499-511, June 2007.