# AN ANALYSIS OF THE DIFFERENT APPROACHES TO NETWORK PACKET CLASSIFICATION

Taufeeq Ahmed[1], Amit kumar[2]
[1]Assistant Professor, Dept of ISE, GNDEC Bidar
[2]Assistant Professor, Dept of CSE, GNDEC Bidar

**Abstract**
**The process of categorizing network packets into flows in a Network Device (Internet router, firewall etc), is called packet classification. All packets belonging to the same flow obey a predefined rule and are processed in a similar manner by the network device. For example, all packets with the same source and destination IP addresses may be defined to form a flow. Packet classification is an enabling function for a variety of Internet applications including Quality of Service, security, monitoring, and multimedia communications. In order to classify a packet as belonging to a particular flow or set of flows, network nodes must perform a search over a set of filters using multiple fields of the packet as the search key.**
**In general, packet classification on multiple fields is a difficult problem. Hence, re- searchers have proposed a variety of algorithms which, broadly speaking, can be categorized as hardware based (architectural) and software based (algorithmic) approaches. [1], [2]**
**The network devices has to do packet classification at line speeds (ex OC standards) etc and given the inability of early algorithms to meet performance constraints imposed by these high speed links, researchers devised hardware based (architectural) solutions to the problem. This thread of research produced the most widely used packet classification de- vice technology, like Ternary Content Addressable Memory (TCAM) etc. But because of the limitations of this approach as pointed out by George Varghese [3], new research is focused more on software based(algorithmic) approach or combination of both.**
**Keywords: Packet Classification, Longest Prefix March (LPM), Bloom Filter, Hash Table, Decision Table, Firewall, Class Bench**.

## Introduction
In this section first we give the definition and importance of packet classification as a general problem and then mention about the metrics of evaluating the performance and also the important fields used for the classification of packets. Then we mention the different applications of packet classification and finally we give an overview the different algorithmic techniques that we have studied.

### 1.1 Definition and Importance
Until recently, network devices provided only best-effort service, servicing packets in a first- come-first-served manner. network devices are now called upon to provide different qualities of service to different applications which means they need new mechanisms such as admission control, resource reservation, per-flow queuing, intrusion detection, fair scheduling etc. All of these mechanisms require the network devices to distinguish packets belonging to different flows. Flows are specified by rules applied to incoming packets. We call a collection of rules a classifier. Each rule specifies a flow that a packet may belong to based on some criteria applied to the packet header, as shown in Figure 1.1 [2].

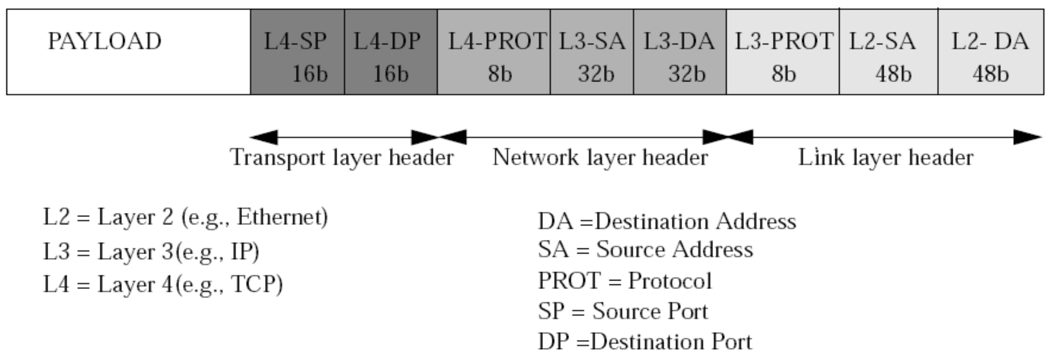


Figure 1.1: Example fields used for classification [ 2]

Although not shown in this figure, higher layer (e.g., application-level) headers may also be used.

## 1.1 Problem statement

A classifier is defined by a set of rules and each rule R of a classifier has d components. R[i] is the ith component of rule R, and is a regular expression on ith the field of the packet header. A packet P is said to match rule R, if for all i, the field of the header of P satisfies the regular expression R[i]. In practice, a rule component is not a general regular expression but is often limited by syntax to a simple address/mask or operator/number(s) specification. In an address/mask specification, a 0 (respectively 1) at bit position x in the mask denotes that the corresponding bit in the address is a don’t care (respectively significant) bit. Examples of operator/number(s) specifications are eq 1232 and range 34-9339. Note that a prefix can be specified as an address/ mask pair where the mask is contiguous i.e., all bits with value 1 appear to the left of bits with value 0 in the mask. It can also be specified as a range of width equal to 2t where t = 32 – prefix length. Most commonly occurring specifications can be represented by ranges. An example real-life classifier in four dimensions is shown in Figure 1.4 [2].

| Rule | Network-layer Destination (address/ mask) | Network-layer Source (address/mask) | Transport-layer Destination | Transport-layer Protocol | Action |
|---|---|---|---|---|---|
| R1 | 152.163.190.69/ 255.255.255.255 | 152.163.80.11/ 255.255.255.255 | * | * | Deny |
| R2 | 152.168.3.0/ 255.255.255.0 | 152.163.200.157/ 255.255.255.255 | eq www | udp | Deny |
| R5 | 152.163.198.4/ 255.255.255.255 | 152.163.160.0/ 255.255.252.0 | gt 1023 | tcp | Permit |
| R6 | 0.0.0.0/0.0.0.0 | 0.0.0.0/0.0.0.0 | * | * | Permit |

Figure 1.4: Example real life classifier [2].

By convention, the first rule R1 is of highest priority and rule R6 is of lowest priority. Some example classification results are shown in Figure 1.5.

| Packet Header | Network-layer Destination | Network-layer Source | Transport-layer Destination | Transport-layer Protocol | Best matching rule, Action |
|---|---|---|---|---|---|
| P1 | 152.163.190.69 | 152.163.80.11 | www | tcp | R1, Deny |
| P2 | 152.168.3.21 | 152.163.200.157 | www | udp | R2, Deny |
| P3 | 152.168.198.4 | 152.163.160.10 | 1024 | tcp | R5, Permit |

Figure 1.5: Example classification results

Longest prefix matching for routing lookups is a special-case of one-dimensional packet classification. All packets destined to the set of addresses described by a common prefix may be considered to be part of the same flow. The address of the next hop where the packet should be forwarded to is the associated action. Here the length of the prefix defines the priority of the rule. A rule which matches a longer prefix of the packet header has higher priority.

1.1.1 Fields used for classification

As we got a general idea of the packet classification problem, below we mention about the general fields (dimensions) used for this problem. The classification of packets can be based on the packet fields like

• Source IP address
• Destination IP address
• Source Destination Port
• Destination Port
• Protocol
• Hardware links

A combination of the above fields or any additional field required depending on the appli- cation may also be used. As mentioned in Section 1.2 a packet P is said to match rule R, if for all i(dimensions), the field of the header of packet P matches with R[i] of a rule R.

### 1.1.2Performance metrics for classification algorithms

The packet classification problem which consists of tens of thousands of rules and also which is done by network devices like Routers(Qi's ), NAP(VPN ), Firewalls (intrusion detection ) etc, has to be done at line speeds like E1, T1, XDSL, OC standards for twisted pair, coaxial cables, optical fibers etc, requires high performance in terms of speed, storage, updates etc. Below we mention a few metrics which are used for the evaluation of the algorithms.

• Search speed - Faster links require faster classification. For example, links running at 10Gbps can bring 31.25 million packets per second (assuming minimum sized 40 byte TCP/IP packets).

• Low storage requirements - Small storage requirements enable the use of fast memory technologies like SRAM (Static Random Access Memory). SRAM can be used as an on-chip cache by a software algorithm and as on-chip SRAM for a hardware algorithm.

• Ability to handle large real-life classifiers.

•

Fast updates - As the classifier changes, the data structure needs to be updated. We can categorize data structures into those which can add or delete entries incrementally, and those which need to be reconstructed from scratch each time the classifier changes. When the data structure is reconstructed from scratch, we call it preprocessing. The update rate differs among different applications: a very low update rate may be sufficient in firewalls where entries are added manually or infrequently, whereas a router with per-flow queues may require very frequent updates.

• Scalability in the number of header fields used for classification.

• Flexibility in specification - A classification algorithm should support general rules, including prefixes, operators (range, less than, greater than, equal to, etc.) And wildcards. In some applications, non-contiguous masks may be required

2.1 Organization of Algorithms

After an extensive study of various research papers we have broadly classified the different algorithmic techniques into the different methods where we study and analyze one or two important techniques from each of the methods. The methods and the techniques are classified as follows

• Decision-tree Based - Modular Packet Classification (Woo)

• Decomposition based - RFC (Recursive flow classification)

• Hash Based - Tuple Space search (TSS), Bloom filter based

First we discuss about the Decision-tree based algorithms where we study

Modular packet Classification proposed by Woo [4]. This algorithm is motivated by intuitive observation on the classification process, and is based on an efficient divide- and-conquer approach. Specifically, we break up the classification procedure into two main steps. In the first step, our algorithm tries to eliminate as many filters as possible by examining specific bit positions. In the second step, the filter bucket is processed to find a match. In essence, the algorithm is a modular composition of two procedures: the first to decompose large filter table into small filter buckets of a fixed maximum size (from 8 to 128), and the second to process filter buckets of limited size to find a match.

Then we discuss about the Decomposition based algorithms where we discuss about RFC(Recursive flow classification) [5], which performs the parallel lookups on each individual fields(dimensions) first (phase 0), then combines the results from the previous phase to form the keys for the next phase while reducing the dimensions of search in each phase gradually.

Finally we deal with Hash based algorithms, where we look at Tuple Space Search [6], [7] which divides the filter set into different groups based on the key lengths formed by combining the significant bits of each dimension and Bloom filter [10], [13] based approach which utilizes a combination of hash functions and a bit vector to store and retrieve information.

We deal with a detailed explanation of the Algorithms with the Implementation details in the next chapter.

## Implementation and Results

First we mention about the Modular Packet Classification as a technique, which follows the Decision tree approach to packet classification and do its analysis. Then we mention about the Recursive Flow classification as a technique, which follows the Decomposition approach to packet classification and do its analysis. Finally we mention about the Tuple space Search and Bloom filter based techniques, which follows the Hashing approach to packet classification and do its analysis.

### 3.0.1 Metrics for evaluation

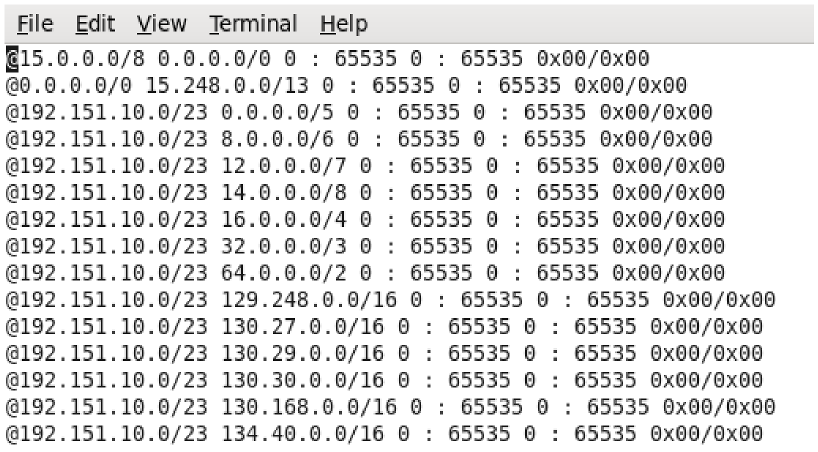
We measure the space efficiency of an algorithm using the average number of bytes con- sumed per filter. We measure the throughput of an algorithm using the memory bandwidth consumed. The number of bytes per memory access and the number of dependent memory accesses per packet lookup. The memory bandwidth consumption is evaluated in both the worst case and the average case. The overall data structure size is the product of the memory consumption per filter and the number of filters. The overall throughput can be calculated by dividing the total memory bandwidth by the memory bandwidth consumed per packet lookup.

### 3.0.2 Usage of filter Set

The open-source ClassBench [15], [16] is used to generate synthetic filter sets with different scales and structures. We provide the parameters used for filter set generation. We also generate a packet header trace using ClassBench for each filter set for implementation verification and algorithm evaluation. The size of a trace is about 10 times of that of the corresponding filter set. We provide the original filter sets that are used as seeds for the synthetic filter sets. The statistics files extracted from the original filter sets can be downloaded from the ClassBench website [ 15].

We only consider 5-tuple filters: source IP address, destination IP address, source port, destination port, protocol. The filter format is ”@source IP address prefix in dot-decimal notation/prefix length destination IP address prefix in dot-decimal notation/prefix length low source port : high source port low destination port : high destination port protocol value in hexadecimal/protocol mask in hexadecimal”. The header trace format is ”source IP address in decimal destination IP address in decimal source port value in decimal destination port value in decimal protocol in decimal”

A sample of the filter set rule file is as shown in the below Figure 3.1.



```
@15.0.0.0/8 0.0.0.0/0 0 : 65535 0 : 65535 0x00/0x00
@0.0.0.0/0 15.248.0.0/13 0 : 65535 0 : 65535 0x00/0x00
@192.151.10.0/23 0.0.0.0/5 0 : 65535 0 : 65535 0x00/0x00
@192.151.10.0/23 8.0.0.0/6 0 : 65535 0 : 65535 0x00/0x00
@192.151.10.0/23 12.0.0.0/7 0 : 65535 0 : 65535 0x00/0x00
@192.151.10.0/23 14.0.0.0/8 0 : 65535 0 : 65535 0x00/0x00
@192.151.10.0/23 16.0.0.0/4 0 : 65535 0 : 65535 0x00/0x00
@192.151.10.0/23 32.0.0.0/3 0 : 65535 0 : 65535 0x00/0x00
@192.151.10.0/23 64.0.0.0/2 0 : 65535 0 : 65535 0x00/0x00
@192.151.10.0/23 129.248.0.0/16 0 : 65535 0 : 65535 0x00/0x00
@192.151.10.0/23 130.27.0.0/16 0 : 65535 0 : 65535 0x00/0x00
@192.151.10.0/23 130.29.0.0/16 0 : 65535 0 : 65535 0x00/0x00
@192.151.10.0/23 130.30.0.0/16 0 : 65535 0 : 65535 0x00/0x00
@192.151.10.0/23 130.168.0.0/16 0 : 65535 0 : 65535 0x00/0x00
@192.151.10.0/23 134.40.0.0/16 0 : 65535 0 : 65535 0x00/0x00
```

Figure 3.1: Filter set – Sample Rule file

## 3.1 Modular Packet Classification (Woo's Algorithm)

It is a Decision tree based algorithm, which is an extension to the method HiCuts proposed by Gupta and McKeown [11].The algorithm is motivated by intuitive observation on the classification process, and is based on an efficient divide-and-conquer approach. The details of the algorithm are as mentioned below.

### 3.1.1 Algorithm

Woo's modular packet classification algorithm is a decision tree-based packet classification algorithm. Unlike the other decision tree-based algorithm, it takes a more direct view over the filters. If we see each filter as a ternary bit string and would like to pick some bit to split the filter set. We would like the two resulting subsets to have similar number of filters (maximize "balancedness") and as few filters as possible to be duplicated in the two subsets (minimize duplication). Clearly, if a particular bit position is chosen, all filters with this bit bearing a value '1' go to a subset, all filters with this bit bearing a value '0' go to the other subset, and all filters with this bit being "don't care" go to the both subsets. The decision tree construction algorithm recursively splits the filter set with a bit-chosen preference until all the leaf nodes contain less than k filters, where k is the predefined bucket size. Each internal tree node needs to store a bit position, which takes log2(n) bits if the filters are n-bit long, and two pointers, each pointing to a child node. Each leaf nodes stores a small set of filters. The lookup algorithm is very simple. Based on the pattern of the packet header, the algorithm follows the chosen bits to locate the target leaf node and then performs a linear search on the filter list. The overall algorithm data structure is shown in the following Figure 3.2

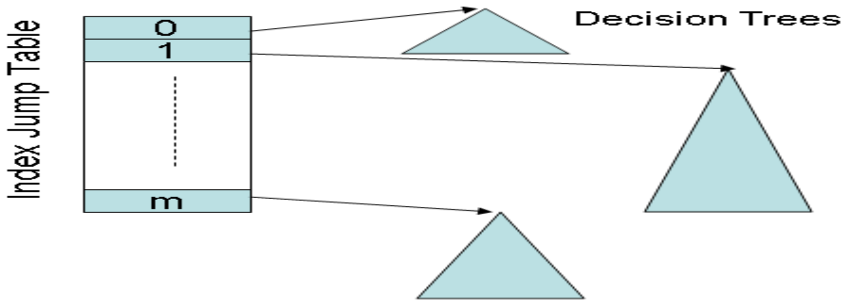


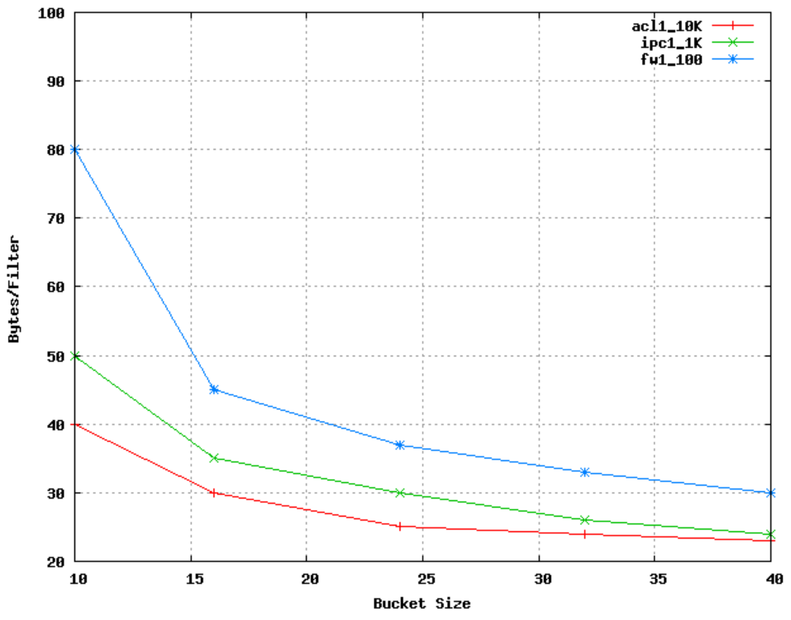
Figure 3.2: Modular packet classification

First, the filters are statically grouped using some prefix bits from some selected header fields. Addressed by an index (which is the numerical value of the concatenated bit string of the chosen prefix bits), each group of filters is then split to generate a decision tree. The preference of bit j is defined as: preference[j] = $(D_j - D_{min})/(D_{max} - D_{min}) + (N*_j - N*_{min})/(N*_{max} - N*_{min})$. In the equation, $D_j = |N0_j - N1_j|$. $N0_j$, $N1_j$, and $N*_j$ are the number of 0, 1, and * ("don't care") at bit position j, respectively. When splitting a filter set, we choose the bit position that results in the least preference value. If there is a tie, we choose the least bit position.

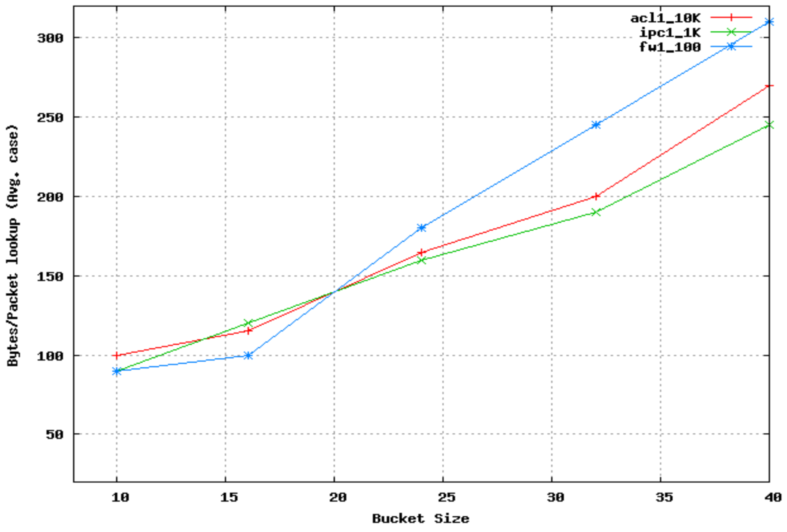
### 3.1.3 Evaluation Results

We measure the space efficiency of an algorithm using the average number of bytes con- sumed per filter. We measure the throughput of an algorithm using the memory bandwidth consumed. The number of bytes per memory access and the number of dependent memory accesses per packet lookup. The memory bandwidth consumption is evaluated in the average case. The overall data structure size is the product of the memory consumption per filter and the number of filters. The overall throughput can be calculated by dividing the total memory bandwidth by the memory bandwidth consumed per packet lookup.

In this simulation, we set the bucket size to 16. No index jump table is used. As shown in the Figure 3.3 below, generally, the ACL1 filter sets demonstrate better performance and scalability and the FW1 filter sets give the poorer overall performance. Before a point, all the three types of filter sets have similar throughput performance but the FW1 filter sets consume significant larger storage. When the number of filters exceeds 5K for FW1, the performance becomes unacceptable, so the data points are not shown in the figure.



(a) The storage requirement for the filter set



(b) The packet classification performance

Figure 3.3: Modular Packet classification – Performance

## 3.2 Recursive Flow Classification (RFC)

The RFC algorithm proposed by Gupta and McKeown [5] is a Decomposition based, multi-stage algorithm. It uses the fact that the classifiers contain considerable structure and redundancy that can be exploited. The details of the algorithm are as mentioned below.

### 3.2.1 Algorithm

The RFC algorithm is a decomposition-based algorithm, which provides very high lookup throughput at the cost of low memory efficiency. It performs the parallel lookups on each individual field first (phase 0). This step is typically conducted by a direct table lookup in order to achieve the best throughput performance. The table size depends on the number of bits of a header field (e.g. A field with n bits requires a table with 2ˆn entries). For memory efficiency, the fields such as the source IP address and the destination IP address are broken into shorter 16-bit chunks and a table is built for each chunk.

The table entry i of chunk j virtually stores the set of filters for which the chunk j covers the value i. actually, in a table, each unique set of filters is binary encoded. The identifier of the encoded set is named eqID. Clearly, for packet classification, the intersection of the sets of filters from all the chunk table lookups is exactly the set of filters that matches a packet. Again, for the best throughput performance, this step is preferred to be conducted from a direct table lookup. This implies we need to build a cross-product table with the number of entries equaling to the multiplication of the number of eqIDs of each chunk. Unfortunately, this number could be terribly large, resulting in unacceptable inefficiency of memory use. For this reason, the cross-producing is conducted recursively in multiple phases.

As a tradeoff of throughput and storage, the number of cross-producing phases can be varied. At one extreme, we can finish the cross-producing in a single step and the algorithm is degenerated into the variation of the naive cross-producing algorithm (2-phase RFC). At the other extreme, we can take up to k phases to finish if there are k chunk tables in total. For 5-tuple packet classification with 7 chunks, the paper proposes to finish the cross-producing in 3 or 4 phases. It is also critical to determine the shape of the reduction tree. A tree is chosen based on two heuristics. 1 - The most correlated chunks (e.g. the two chunks from the source IP address field) should be merged first. 2 - As long as the memory consumption is reasonable, the algorithms should combine as many chunks as possible.

An Example reduction tree is shown in the following Figure 3.4

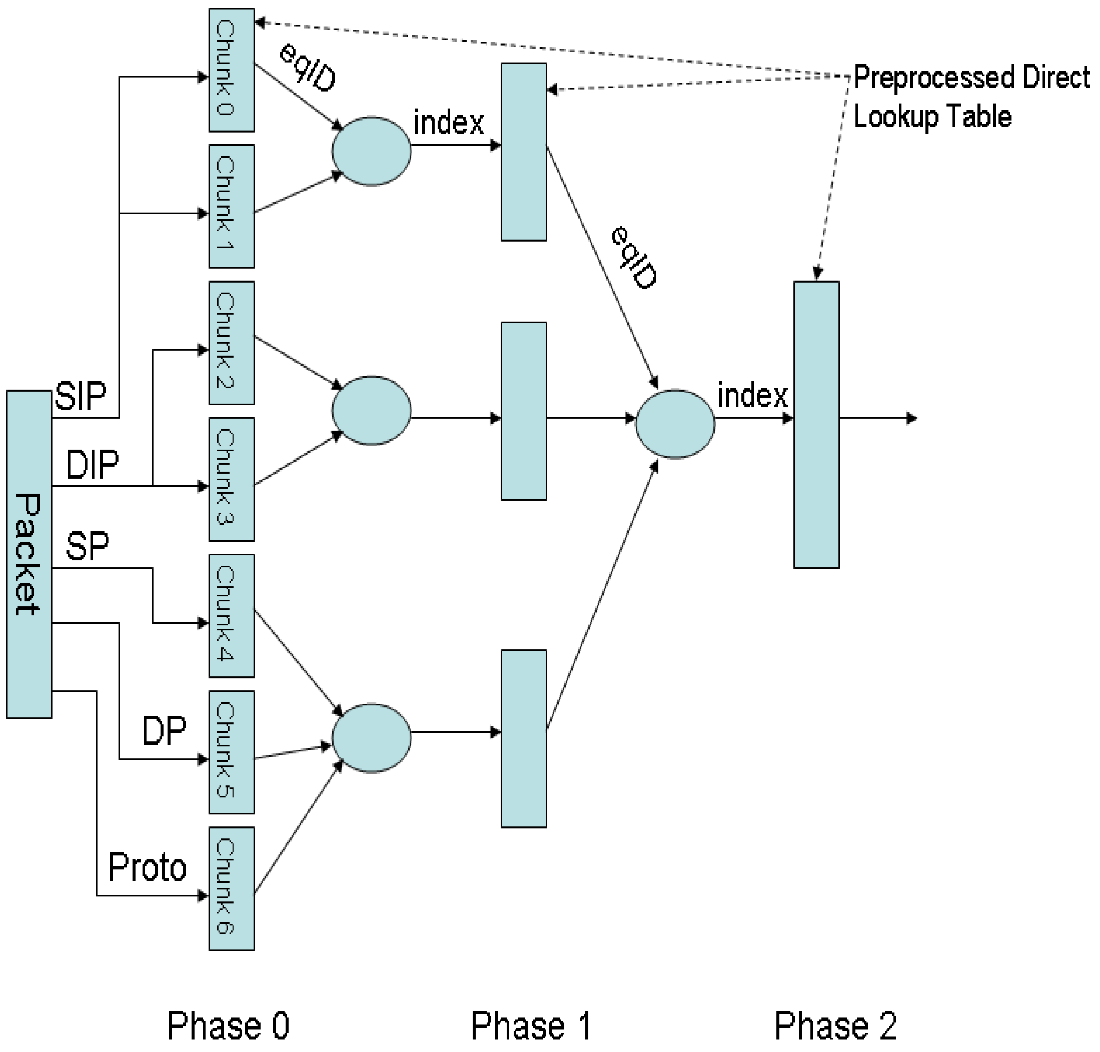


Figure 3.4: Recursive Flow Classification-General Flow

### 3.2.2 Construct the Chunk Lookup Tables

The following figure shows an example of constructing the lookup table. Assuming each chunk is 4-bit long, we draw the chunk x and the chunk y in a 2D plane. Projected to this 2D plane, each filter appears to be a rectangle. We then project the end points of each rectangle to the axis. Any two adjacent projection points on an axis defines an elementary interval which is fully covered by a set of filters. We say an elementary interval represent a set of filters. Two neighboring elementary intervals cannot represent the same filters; whereas two nonadjacent elementary intervals are possible to represent the same filters. We assign each elementary interval an identifier, named eqID. The elementary intervals representing the same set of filters are assigned the same eqID, as shown in the following Figure 3.4

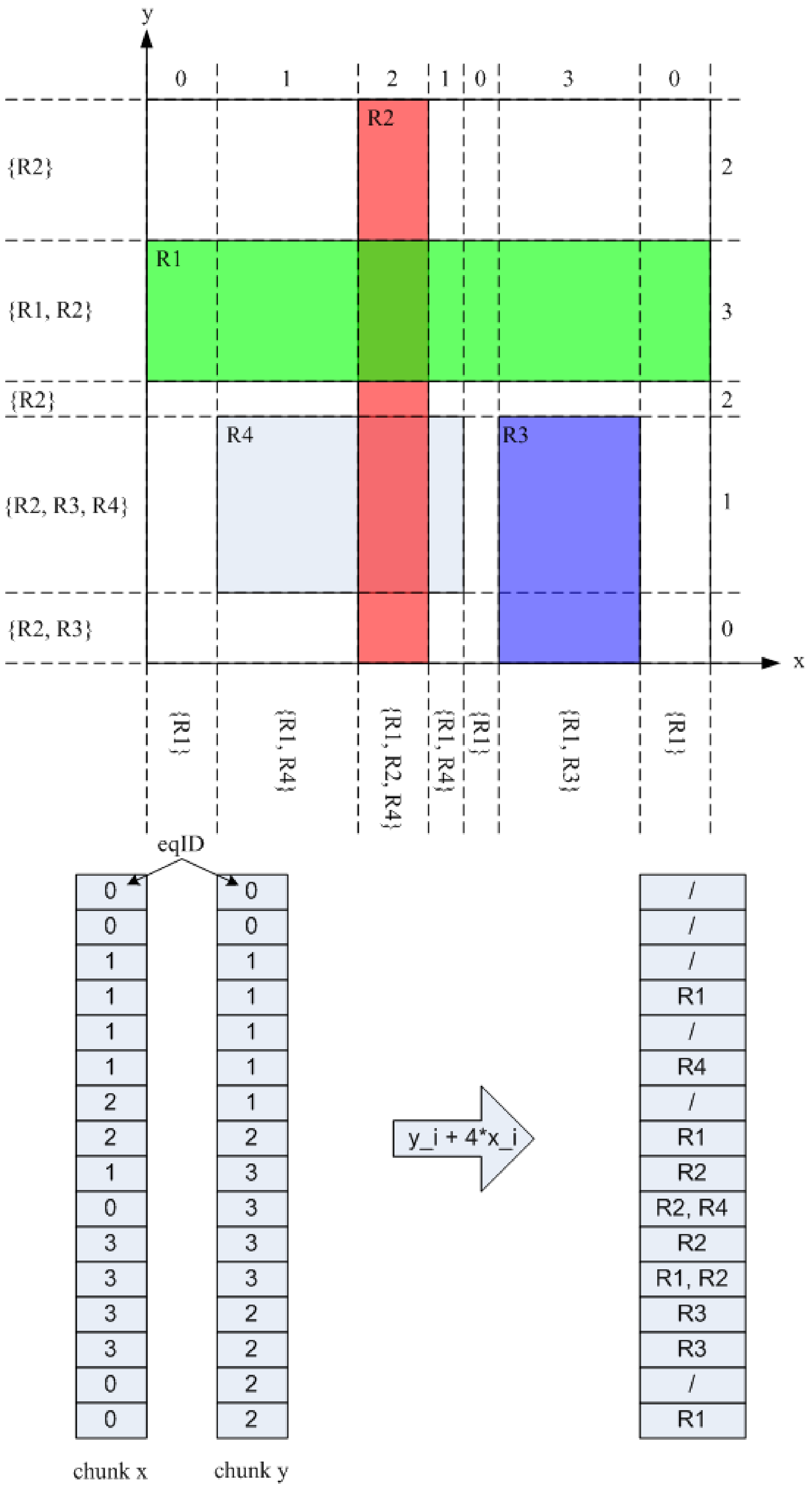
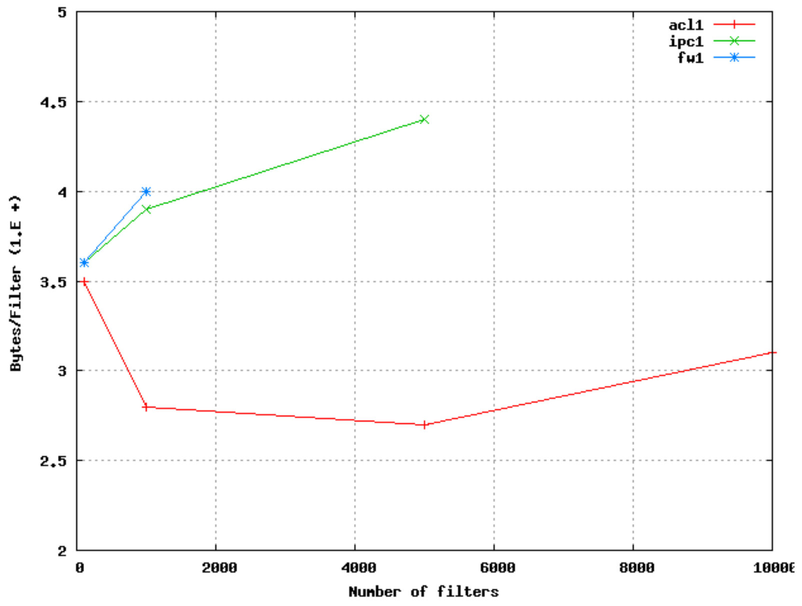


Figure 3.5: Recursive Flow – Classification of Elementary Interval

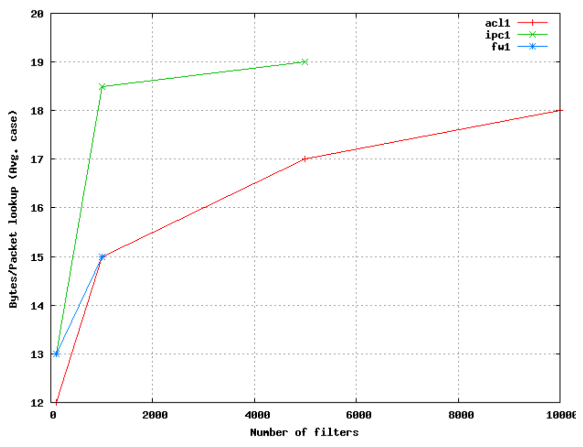
In the above example, chunk x has 4 eqIDs (encoded as 0 to 3) and chunk y has 4 eqIDs too. Since each chunk has 4 bits, the chunk lookup table contains 2ˆ4=16 entries. The table index is simply the numerical value of the chunk bit string. We fill out each table entry with the corresponding eqID. For example, given a chunk x with bit string ”1001”, we find it drop in the elementary interval labeled with eqID 0 (which uniquely represents the filter set R1), so we fill out the entry 9 of the chunk x lookup table with 0. Assume in the second phase, we would like to combine the chunk x and the chunk y. Basically, we need to construct a cross-product table with m*n entries, where m is the number of unique eqIDs for the chunk x and n is the number of unique eqIDs for the chunk y. In this example, the cross-product table has 16 entries. If this is not the last phase, again, each table entry should be filled with an eqID that represents a unique set of filters. In this example, we simply fill out each entry with the set of filters. During the packet classification, if the chunk x table lookup returns an eqID 2 and the chunk y table lookup returns an eqID 3, then the entry index of the cross product table should be 3+4*2=11. We retrieve that entry and know that filter R1 and R2 are matched.

### 3.2.5 Evaluation Results

We measure the space efficiency of an algorithm using the average number of bytes con- sumed per filter. We measure the throughput of an algorithm using the memory bandwidth consumed. The number of bytes per memory access and the number of dependent memory accesses per packet lookup. The memory bandwidth consumption is evaluated in the average case. The overall data structure size is the product of the memory consumption per filter and the number of filters. The overall throughput can be calculated by dividing the total memory bandwidth by the memory bandwidth consumed per packet lookup.



(a) The storage requirement for the filter set



(b) The packet classification performance

Figure 3.7: Recursive Flow classification- Performance

In the above simulation, we use the 4-phase configuration. The lookup throughput gets slightly worse when the filter sets become larger. The memory efficiency does not necessarily get worse when the filter sets become larger, as in the case of ACL1 filter sets. For the IPC1 filter sets with more than 5K filters and the FW1 filter sets with more than 1K filters, the storage becomes unacceptably large

### 3.3 Tuple Space Search - (TSS)

This technique introduced by G.Varghese [ 6], makes use of the fact that the real databases use only a small number of distinct field lengths. Thus by mapping filers to tuples (formed by combination of field lengths) different tuple groups are formed. Finally hashing is used to search for an entry in the tuple group. The details of the algorithm is as below.

#### 3.3.1 Algorithm

The Tuple Space Search algorithm is a hash-based algorithm. A tuple is defined as a vector of k lengths, where k is the number of fields in a filter. For example, in a 5-field filter set, the tuple [7, 12, 8, 0, 16] means the length of the source IP address prefix is 7, the length of the destination IP address prefix is 12, the length of the protocol prefix is 8 (an exact protocol value), the length of the source port prefix is 0 (wildcard or"don't care"), and the length of the destination port prefix is 16 (an exact port value). We can partition the filters in a filter set to the different tuple groups. Since the filters in a same tuple group have the same tuple specification, they are mutual exclusive and none of them overlaps with others in this tuple group. Now we can perform the packet classification across all the tuples to find the best matched filter. If multiple tuple groups report matches, we resolve the best matched filter by comparing their priorities. The filters in a tuple can be easily organized into a hash table, where we use the tuple specification to extract the proper number of bits from each field as the hash key. Assume there is no hash collision in the hash tables. One memory access can determine if there is a matched filter in a hash table so the lookup performance is only determined by the number of tuple specifications. If the hash tables are properly implemented, this algorithm gives an excellent storage performance, which is linear to the filter set size.

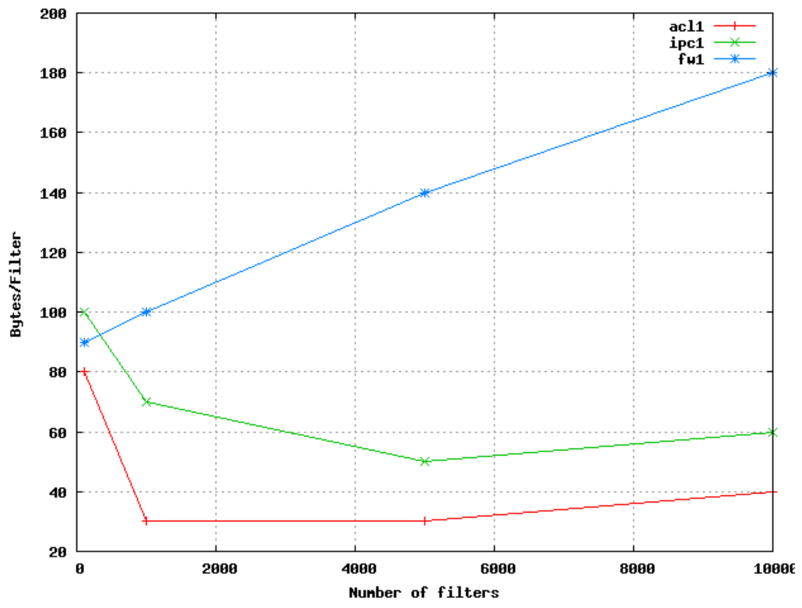
#### 3.3.2 Algorithm Optimization

Tuple pruning since the number of tuples can be very large, the lookup throughput performance suffers. The tuple pruning technique is developed to reduce the number of tuples that has to be searched during the lookups. The observation is for any given packet, the number of unique prefixes matched on a particular field is typically small. So if we could perform the longest prefix match (LPM) first on some field and figure out the lengths of the matched prefixes, then only a subset of tuple groups need to be searched. If LPMs are performed on more additional fields, we can filter out more tuples. Here is a tradeoff: more LPMs mean larger storage and at the same time, more single field lookups, which at some point may even lower the overall lookup throughput. In practice, the LPMs performed on only the source IP address field and the destination IP address field can achieve the significant tuple reduction at the reasonable extra cost.

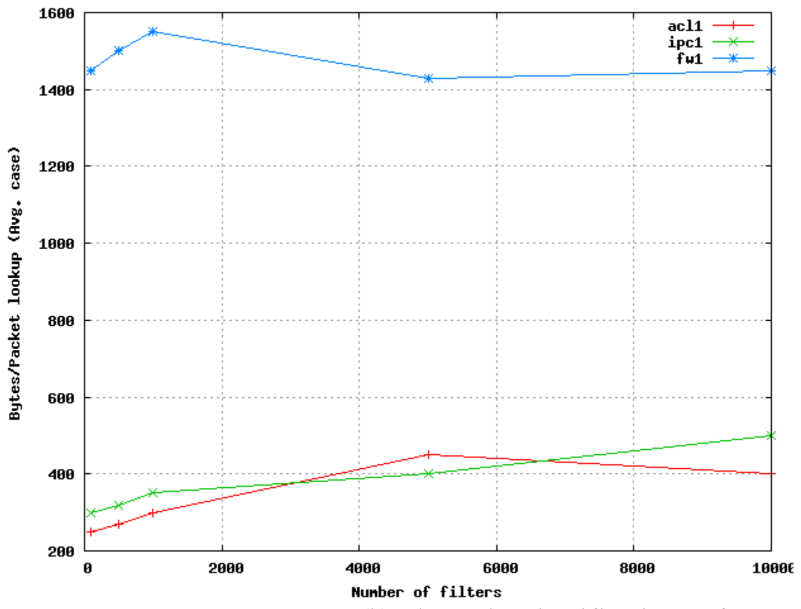
### 3.3.5 Evaluation Results

We measure the space efficiency of an algorithm using the average number of bytes con- sumed per filter. We measure the throughput of an algorithm using the memory bandwidth consumed. The number of bytes per memory access and the number of dependent memory accesses per packet lookup. The memory bandwidth consumption is evaluated in both the worst case and the average case. The overall data structure size is the product of the memory consumption per filter and the number of filters. The overall throughput can be calculated by dividing the total memory bandwidth by the memory bandwidth consumed per packet lookup.

The algorithm works better on the ACL1 and the IPC1 filter sets. It also shows the best storage and throughput scalability on these two types of filter sets.



(a) The storage requirement for the filter set



(b) The packet classification performance

Figure 3.9: Tuple space search- Performance

## 3.4 Bloom filter based

A Bloom filter is essentially a bit-vector of length m used to efficiently represent a set of messages. Given a set of messages X with n members, the Bloom filter is programmed as follows. For each message x in X, k hash functions are computed on x producing k values each ranging from 1 to m. Each of these values address a single bit in the m-bit vector, hence each message x causes k bits in the m-bit vector to be set to 1. Note that if one of the k hash values addresses a bit that is already set to 1, that bit is not changed. Querying the filter for set membership of a given message x is similar to the programming process. Given message x, k hash values are generated using the same hash functions used to program the filter. The bits in the m-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 message is declared to be a non-member of the set. If all the bits are found to be 1, then the message 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 X, then it is said to be a false positive.

Let k be the number of hash functions used, m be the size of the bit vector and n be the number of elements inserted then the false positive probability f, is given by the Equation 3.1

$$f = (1 - (1 - 1/m)^{nk})^{k} \qquad (3.1)$$

For large values of f the above equation reduces to

$$f \approx \left(1 - {}^{\frac{-nk}{m}}\right)^{k} \qquad (3.2)$$

The false positive probability at this optimal point is given by

$$f = \frac{1}{2})^{k} \qquad (3.3)$$

It should be noted that if the false positive probability is to fixed, the the size of the filter m needs to scale linearly with the size of the message set, n.

### 3.4.1 Basic Configuration

A basic configuration of our approach is shown in Figure 3.10. We begin by grouping the database of prefixes into sets according to prefix length. The system employs a set of W counting Bloom filters where W is the length of input addresses, and associates one filter with each unique prefix length. Each filter is programmed" with the associated set of prefixes according to the previously described procedure. A hash table is also constructed for each distinct prefix length. Each hash table is initialized with the set of corresponding prefixes, where each hash entry is a [prefix, next hop] pair. While we assume the result of a match is the next hop for the packet, more elaborate information may be associated with each prefix if so desired.

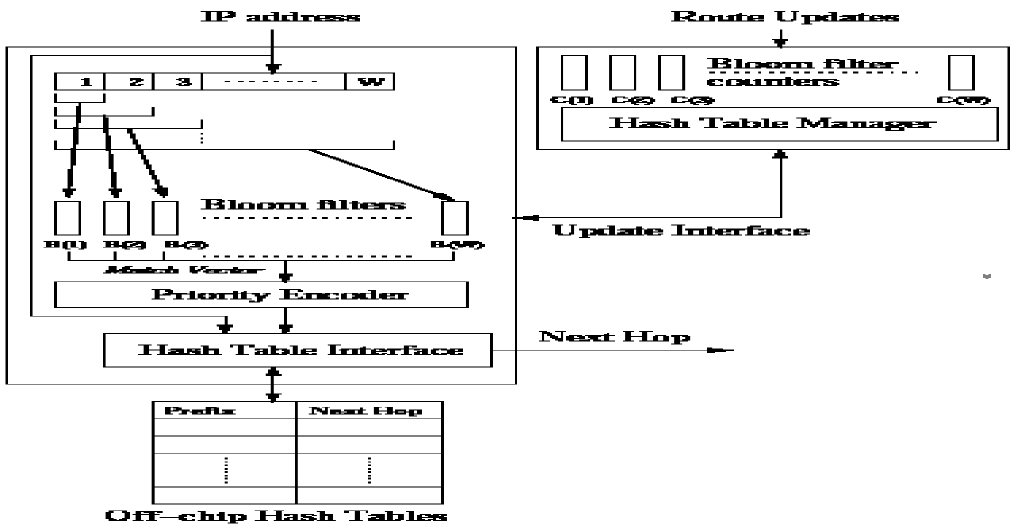


Figure 3.10: Basic Configuration of Longest prefix matching using Bloom filters [ 10]

A search proceeds as follows. The input IP address is used to probe the set of W Bloom filters in parallel. One-bit prefix of the address is used to probe the filter associated with length one prefixes, two-bit prefix of the address is used to probe the filter associated with length two prefixes, etc. Each filter simply indicates match or no match. By examining the outputs of all filters, we compose a vector of potentially matching prefix lengths for the given address, which we will refer to as the match vector. Consider an IPv4 example where the input address produces matches in the Bloom filters associated with prefix lengths 8, 17, 23, and 30; the resulting match vector would be f8, 17, 23,30g. The search proceeds by probing the hash tables associated with the prefix lengths represented in the match vector in order of longest prefix to shortest. The search continues until a match is found or the vector is exhausted.

### 3.4.2 Optimization

To reduce the number of Bloom filters prefix distribution of the IPv4 is used as a heuristic, which shows a large numbers of 24-bit prefixes and few prefixes of length less than 8-bits. An average prefix distribution for all of the tables we collected is shown in Figure 3.11

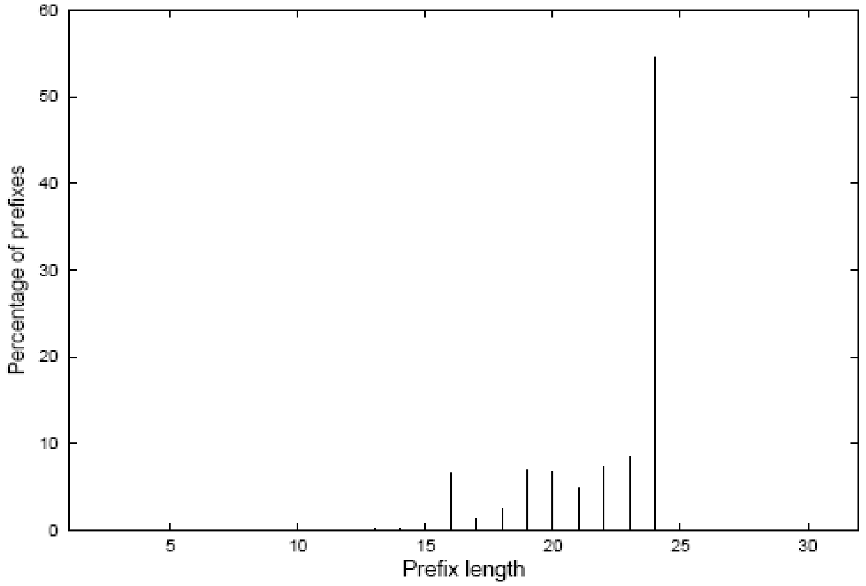


Figure 3.11: Average prefix length distribution for IPv4 BGP table [10]

Several variables affect system performance and resource utilization as below

• N, the target amount of prefixes supported by the system

• M, the total amount of embedded memory available for Bloom filters

• $W_{dist.}$, the number of unique prefix lengths supported by the system
• $m_i$, the size of each Bloom filter
• $k_i$, the number of hash functions computed in each Bloom filter
• $n_i$, the number of prefixes stored in each Bloom filter

Reducing the Number of Filters

The following methods are used to reduce the number of Bloom filter

Direct lookup array: We observe from the distribution statistics that sets associated with the first few prefix lengths are typically empty and the first few non-empty sets hold few prefixes as shown in Figure 2. This suggests that utilizing a direct lookup array for the first a prefix lengths is an efficient way to represent shorter prefixes while reducing the number of Bloom filters. For every prefix length we represent in the direct lookup array, the number of worst case hash probes is reduced by one. Use of a direct lookup array also reduces the amount of embedded memory required to achieve optimal average performance, as the number of prefixes represented by Bloom filters is decreased. CPE : We can reduce the number of remaining Bloom filters by limiting the number of distinct prefix lengths via further use of Controlled Prefix Expansion (CPE). Clearly, the appropriate choice of CPE strides depends on the prefix distribution. As illustrated in the average distribution of IPv4 prefixes shown in Figure 2, we observe in all of our sample databases that there is a significant concentration of prefixes from lengths 21 to 24. On average, 75.2% of the N prefixes fall in the range of 21 to 24. Likewise, we observe in all of our sample databases that prefixes in the 25 to 32 range are extremely sparse. Specifically, 0.2% of the N prefixes fall in the range 25 to 32.

Now the expression for the expected number of hash probes per lookup becomes

$$E_{exp} = 2 \times (\frac{1}{2})^{\frac{M \ln 2}{\alpha_{24}N_{24}+\alpha_{32}N_{32}}} + 1 \qquad (3.4)$$

### 3.4.3 Evaluation Results

We measure the space efficiency of an algorithm using the average number of bytes con- sumed per filter. We measure the throughput of an algorithm using the memory bandwidth consumed. The number of bytes per memory access and the number of dependent memory accesses per packet lookup. The memory bandwidth consumption is evaluated in both the worst case and the average case. The overall data structure size is the product of the memory consumption per filter and the number of filters. The overall throughput can be calculated by dividing the total memory bandwidth by the memory bandwidth consumed per packet lookup. As can been seen from the Figure 3.12 below as the size of the Embedded memory is increased the the number of hash probes required reduces to nearly 1.

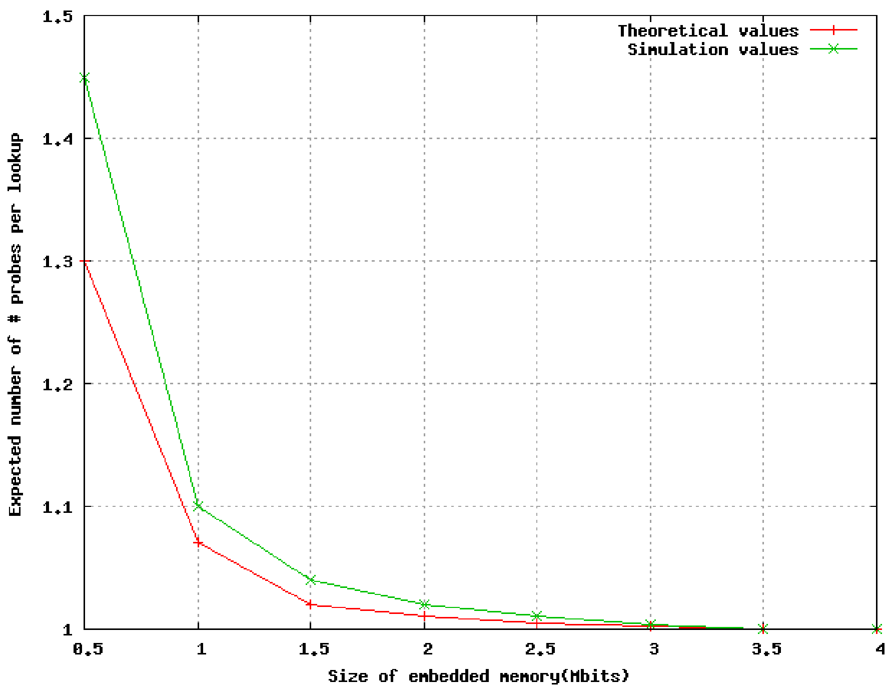


Figure 3.12: Average number of hash probes per lookup for various embedded memory sizes

## Conclusionand Future Work

### 4.1 Conclusion

We see that the Modular packet classification method by Woo [4] combines a heuristic tree search for separating a large set of filters into fixed size filter buckets, and another search procedure for searching through fixed size filter buckets and it is motivated by the practical observation that a single search approach may not be optimal for filter table of all sizes.

Recursive flow classification techniques relies on the fact that real classifiers exhibit a considerable amount of structure and redundancy, and introduces for the idea of using simple heuristic algorithms to solve the multi-dimensional packet classification problem. RFC deliberately attempts to exploit this structure in the real classifiers and also extends the idea of cross-production technique.

Tuple space search algorithm searches through the field length combinations found in the filter set. It is motivated by the observation that the number of tuples in real databases is much smaller than the number of filters. The algorithm works better on the ACL1 and the IPC1 filter sets. It also shows the best storage and throughput scalability on these two types of filter sets as shown in Figures 3.9a and 3.9b.The tuple pruning optimization has the significantly effect on reducing the number of tuples that need to be searched.

Here we have used Bloom filter based approach only for the Longest Prefix Match prob- lem, though it can be easily scaled to packet classification of packet in multiple dimensions as demonstrated by Dharmapurikar [12]. For the LPM case we find that the as the size of the Embedded memory increases the throughput for finding a match reduces to 1, which is one of the best algorithmic approach in terms of throughput as shown in the Figure 3.12.

Bloom filter based approach which uses Hashing as its core approach, is a generic tech- nique which can be applied to varied fields. To use it with packet classification usually we use it in conjunction with the existing techniques, which greatly reduces the search time and provides high efficiency. It can be combined with the existing algorithmic techniques and can easily be scaled to the hardware as well [10]. Its usage with the existing cross-production algorithm is well demonstrated by the paper [12] which also employs the algorithmic approach but it can be easily scaled to the hardware level as well.

## References

[1] David E. Taylor Survey & Taxonomy of Packet Classification Techniques, ACM COM- PUTING SURVEYS, May. 2004.

[2] Pankaj Gupta, Nick McKeown Algorithms for Packet Classification, Proc IEEE Network Special Issue, vol. 15, no. 2, march 2001.

[3] Florin Baboescu, Sumeet Singh, George Varghese Packet Classification for Core Routers: Is there an alternative to CAMs? Proc IEEE INFOCOM 2003.

[4] T. Woo, A modular approach to packet classification: Algorithms and results, in INFO- COM, 2000.

[5] Pankaj Gupta, Nick McKeown Packet classification on multiple fields, Proc ACM SIG- COMM '99, Volume 29 Issue 4, Oct. 1999

[6] V. Srinivasan, Subhash Suri, and George Varghese. Packet classification using tuple space search In Proc of ACM-SIGCOMM, pages 135146, Cambridge, Massachusetts, USA, September 1999.

[7] Hyesook Lim, So Yeon Kim Tuple Pruning Using Bloom Filters for Packet Classification, IEEE Micro archive, Volume 30 Issue 3, May 2010

[8] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner. Scalable high speed IP routing table lookups. In Proc of ACM SIGCOMM, pages 2536, September 1997.

[9] George Varghese. Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices, 2005.

[10] S. Dharmapurikar, P. Krishnamurthy, and D.E. Taylor, Longest Prefix Matching Using Bloom Filters, IEEE/ACM Trans. Networking, vol. 14, no. 2, pp. 397-409, Apr. 2006.

[11] P. Gupta and N. McKeown, Packet Classification using Hierarchical Intelligent Cut- tings, Hot Interconnects VII, August 1999.