Hindawi Publishing Corporation
Journal of Computer Systems, Networks, and Communications
Volume 2008, Article ID 603860, 14 pages
doi:10.1155/2008/603860

Research Article

Packet Classification by Multilevel Cutting of the Classification Space: An Algorithmic-Architectural Solution for IP Packet Classification in Next Generation Networks

Motasem Aldiab, Emi Garcia-Palacios, Danny Crookes, and Sakir Sezer

Institute of Electronics, Communications and Information Technology, Queen's University Belfast, Northern Ireland Science Park, Belfast BT3 9DT, UK

Correspondence should be addressed to Motasem Aldiab, maldiab01@qub.ac.uk

Received 2 May 2008; Accepted 14 September 2008

Recommended by Maode Ma

Traditionally, the Internet provides only a "best-effort" service, treating all packets going to the same destination equally. However, providing differentiated services for different users based on their quality requirements is increasingly becoming a demanding issue. For this, routers need to have the capability to distinguish and isolate traffic belonging to different flows. This ability to determine the flow each packet belongs to is called packet classification. Technology vendors are reluctant to support algorithmic solutions for classification due to their nondeterministic performance. Although content addressable memories (CAMs) are favoured by technology vendors due to their deterministic high-lookup rates, they suffer from the problems of high-power consumption and high-silicon cost. This paper provides a new algorithmic-architectural solution for packet classification that mixes CAMs with algorithms based on multilevel cutting of the classification space into smaller spaces. The provided solution utilizes the geometrical distribution of rules in the classification space. It provides the deterministic performance of CAMs, support for dynamic updates, and added flexibility for system designers.

1. INTRODUCTION

Traditionally, the Internet provides only a "best-effort" service, treating all packets going to the same destination similarly. However, providing differentiated services for different users based on their quality requirements is increasingly becoming a demanding issue. For this, routers need to have the capability to distinguish and isolate traffic belonging to different flows. This ability to determine the flow each packet belongs to is called packet classification.

Originally, classification came from routing, where the route lookup problem could be seen as a special case of the more general packet classification problem. The primary task of routers is to forward packets from input links to the appropriate output links. In order to do this, Internet routers must consult a route table containing a set of addresses and the output link or next hop for packets destined for each address. The task of resolving the next hop from the destination IP address is commonly referred to as route lookup or IP lookup. The adoption of classless interdomain routing (CIDR) allows the network addresses in route tables to be of any size. This makes entries in the lookup table of variable-length, and subsequently, a route lookup requires finding the longest matching prefix in the table for the given destination address. As packet classification emerged from the router's need to classify traffic into different flows, the longest prefix matching techniques used in routing lookup tables formed the origin of packet classification techniques.

If an Internet router is to provide more advanced services than packet forwarding, it must perform flow identification. The process of identifying the packets belonging to a specific flow or group of flows between a source and destination is the task of packet classifiers. Packet classification is searching a table of rules for the highest priority rule or set of rules which match the packet. An example of rules in a 5-tuple classifier

is shown in Table 1. The route lookup problem may be seen as a special case of the more general packet classification problem. Applications for quality of service (QoS), security, monitoring, and multimedia communication typically operate on flows, so each packet must be classified in order to assign a flow identifier. Packet classification is needed to facilitate the following: filtering packets for security reasons (Firewalls), delivering packets within certain delay bounds (QoS), enabling premium services, policy-based routing, traffic rate-limiting and policing, traffic shaping, and billing.

In the near future, 40 gigabit per second (OC768) line speeds are expected to be achieved. Given the smallest packet size of 40 bytes in the worst case, the router needs to lookup packets at a speed of 125 million packets per second. Thus, hardware accelerated architectures are inevitable for next generation network classifiers. Technology vendors are reluctant to support algorithmic solutions that take advantage of the availability of cheap commodity RAM due to their nondeterministic performance [1]. Despite the disadvantages of CAMs like silicon cost and high-power consumption, they are favoured by technology vendors for their deterministic performance. CAMs guarantee that packets will be classified at a fixed rate which is necessary to provide services at line speed. Algorithmic-architectural solutions that combine both CAMs and algorithms can improve the efficiency, scalability of the CAMs without sacrificing deterministic high performance.

In this paper, a new architecture that takes advantage of the geometrical characteristics of rule sets, HyperCuts packet classification algorithm, and CAMs is proposed. The proposed architecture works by multilevel cutting of the classification space and it has the deterministic performance of CAMs. As technology providers are reluctant to support algorithmic solutions due to their nondeterministic performance, the proposed architecture is expected to take algorithmic solutions of packet classification based on cutting the classification space from theory to implementation on emerging technologies such as ASIC and FPGAs.

Section 2 of this paper presents the related research and motivation. In Section 3, the packet classification problem is presented from a geometrical point of view. Section 4 presents the proposed architecture. Section 5 provides a detailed analysis of the proposed architecture. Section 6 discusses issues related to the dynamic updates of the architecture, and finally, Section 7 concludes this paper.

2. RELATED RESEARCH AND MOTIVATION

Packet classification solutions are divided mainly into three categories: algorithmic solutions, content addressable memories (CAMs), and algorithmic-architectural solutions.

2.1. Algorithmic solutions

Algorithmic solutions for packet classification take advantage of the availability of cheap commodity memories and they fall into one of the following categories.

2.1.1. Cutting and tries

This is where the classification space is divided into subspaces by forming a decision tree in order to eliminate irrelevant rules until a leaf node is reached which contains the matching rule(s). A description of the algorithms that fall into this category follows.

HiCuts [2] divides the search space into subspaces by cutting across a selected dimension (i.e., source address, destination address, etc.) at each node until the number of rules in the node falls below a predetermined threshold. The selection of the dimension to cut across and the number of cuts is determined by the heuristics of the algorithm.

HyperCuts [3] exploits the idea of HiCuts in dividing the classification space; however, each node in HiCuts represents a hyperplane, while it represents a k-dimensional hypercube in HyperCuts. The aim of that is to reduce the depth of the tree; however, this will make searching at nodes in HyperCuts more complicated.

Modular packet classification [4] organizes the classification space into 3 layers: index jump table, search tree, and rules buckets. As shown in Figure 4, the index jump table divides rules into different groups using some initial prefixes of selected dimensions. The search tree is built by examining a certain number of bits of the rules at a time. The particular bits chosen are any arbitrary unexamined number of bits selected in order to make the tree as balanced as possible and to decrease replication of rules in buckets.

Set pruning tries [5] are a 2-dimensional classification algorithm that builds a trie for destination address prefixes. For each prefix in a destination address trie, there is an associated trie for the relevant source address prefixes. This algorithm works by finding the longest match in the destination trie for the incoming packet header destination address field, then searching the associated source trie to find the matching rule.

An FIS tree [6] is a tree that stores a set of segments or ranges. The leaf nodes of the tree correspond to the elementary intervals on the axis. Projections of the d-dimensional rectangles specified by the rules define elementary intervals on the axes; in this case, elementary intervals are formed on the port axis. N rules will define a maximum of $I = (2N + 1)$ elementary intervals on each axis.

2.1.2. Disintegration

This is where the multiple field search problem is disintegrated (decomposed or disassembled) into instances of the single field search problem. Results are then aggregated either in one stage or several stages. A description of the algorithms that fall into this category follows.

Crossproducting [7] divides the search process into separate search processes—one for each field. For example, if we have 5-tuple rules, then the unique values of each field will be combined in a separate list. For an incoming packet, a search process will be carried out for each field in the corresponding list then the results will be combined in one step to find a match. Crossproducting can provide

Table 1: Rules table of a 5-tuple classifier.

	Source address	Destination address	Source port	Destination port	Protocol	Flow ID
Rule 1	100.1.20.122/23	230.11.123.145/23	100–1024	∗	TCP	ID 1
Rule 2	120.0.30.1/20	147.77.20.54/21	100–1024	100–1024	UDP	ID 2
Rule 3	107.3.44.110/24	115.112.101.2/21	∗	1024–65535	TCP	ID 3
...	...	...	...	...	...	...
Rule N	155.11.33.99/16	133.10.212.134/20	∗	∗	∗	ID N

high-search speed with parallel implementation. However, it suffers from exponential memory requirements.

In the recursive flow classification (RFC) algorithm [8], the problem of packet classification can be considered as a mapping process, mapping S packet header bits onto T flow ID bits. The mapping process is determined by R classifier rules represented by $T = \log_2(R)$ flow ID bits. The lookup operation is performed over several stages by decomposing the lookup field into smaller segments. Each segment is used in parallel to lookup an index that is significantly smaller than the initial segment, compressing the TCP/IP field into a smaller field. Over multiple stages, the smaller segment lookups are then combined, yielding the final lookup value.

DCFL [9] was motivated by two observations on the real rule sets.

(i) Match Condition Redundancy: for each rule field, the number of unique match conditions specified by rules in the rule set is much less than the number of rules in the rule set.

(ii) Match Set Confinement: for each rule field, the number of unique match conditions that can be matched by a single packet header field is small even for larger rule set.

DCFL uses independent search engines for each rule field and then aggregates the results of each field search using a high level of parallelism. DCFL works by labelling unique field values with unique values and assigning a count value for the number of rules specifying the field value. The count values are updated as rules specifying the corresponding values are added or removed. The data structure needs to be updated when the count value changes from 0 to 1 or from 1 to 0. By concatenating, the label of each field value, a unique value for each rule can be obtained.

Packet classification by bit vectors (BVs) [10] is a classification solution that is based on defining elementary intervals on the axes by projecting from the edges of the d-dimensional rectangles specified by the rules. N rules will create at most $2N+1$ elementary intervals on each axis. For each elementary interval on each axis, there is a corresponding N-bit vector. Each bit position corresponds to a rule in the set of rules. The bit positions corresponding to the rules that overlap the associated elementary interval are set to 1, otherwise to 0. For each dimension d, an independent data structure is built to find the elementary interval corresponding to a given point. Baboescu and Varghese [11] utilized the fact that the maximum number of rules matching a packet is limited in real rule sets, so the bit vectors are sparse. They introduced the aggregated bit vector (ABV) algorithm as an enhancement of BV. ABV divides the N-bit vectors into A chunks. Each chunk is N/A bits in size. Each chunk has an associated bit in an A-bit aggregate bit vector.

2.1.3. Arranging into tuples

This is where a tuple defines the number of bits in each field of the rules. Hash functions are used to find the matching rules for each set of rules arranged in a tuple. Classification by arranging into tuples [12, 13] came from the observation that the number of tuples is much less than the number of rules. A tuple defines the number of bits in each field of the rules. As all rules that map to a particular tuple have the same mask, then the required number of bits could be concatenated to construct a hash key from an incoming packet to probe a certain tuple. An exact match technique, such as hash table, could be used to find the matching rule.

2.2. Content addressable memory

CAM [14–18] is a storage array designed to find the location of a particular stored value by comparing the input against all of the values in the storage array simultaneously in 1-clock cycle. In binary CAMs, data comprises only from "1" or "0", while in ternary CAMs (TCAM), data comprises from "1", "0", or "∗" (do not care) bits. In RAM, each bit is stored in a cell, while in CAM, each bit requires comparison circuitry in addition to the storage cell. This makes CAM very expensive in terms of silicon cost and power dissipation in comparison to RAM.

Similar to RAM, CAM writes data in words in the storage array but it reads in a different way. In RAM, the data in a specific location (i.e., the content) is read by inputting the address of that location. In CAM, the input to the array is the data to be searched for (i.e., the content), and the output is the location of the match.

In a binary CAM, the content can be made up of bits comprising two states, "0" and "1". In a ternary CAM, a third "does not care" state can also be included. A TCAM stores content as a (value, mask) pair, where value and mask are each W-bit numbers, and W is the width of the value. In addition to the storage cells that are required for value, a similar number of storage cells are required for the mask. Moreover, the matching circuitry is more complicated than for a binary CAM.

TCAM has the advantage of finding a match in 1 clock cycle; however, they suffer the disadvantages of high-silicon cost and high-power consumption. Typically, a TCAM cell

requires six transistors to store one bit, and the same number of transistors to store the mask bit and four transistors for the match logic. Thus, each TCAM cell requires 16 transistors, which makes the cell 2.7 times larger than the standard SRAM cell [19]. In some architectures, a TCAM cell requires 14 transistors [20, 21]. The power consumption for a single TCAM bit is around 3 microwatts per bit [22], compared with 20–30 nanowatts per bit in SRAM (i.e., a factor of ≈100) [23].

In addition to the above mentioned disadvantages, TCAM suffers storage inefficiency when dealing with ranges. Range comparison is required for port numbers. As TCAM does not store ranges, ranges must be converted to prefixes. For example, the range 1–13 (4 bits) will be represented by the following prefixes: 0001, 001*, 01**, 10**, and 110*. The range to prefix expansion might result in an expansion factor of $2(W-1)$, where W is the field width in bits, that is, 30 for each port field. For the source and destination ports in IPv4, the expansion factor might reach 900 in the worst case.

Multimatch classification is required for new network applications such as intruder detection systems. In [24], the authors presented a solution that produces multimatch classification results with only one TCAM lookup and one SRAM lookup per packet.

In [25], a distributed TCAM scheme that exploits chip-level-parallelism is proposed to improve the packet classification throughput. This scheme seamlessly integrates with a range encoding scheme, which solves the range matching problem and ensures a balanced high-throughput performance.

2.3. Algorithmic-architectural solutions

These solutions mix algorithms with CAMs in order to achieve a tradeoff between the expensive deterministic performance of CAMs and the nondeterministic performance of algorithms based on cheap commodity RAMs. Under this category, there are two solutions: parallel packet classification (P2C) and label encoded CAM (LECAM).

Parallel packet classification (P2C) [26] exploits parallelism between independent field searches and then encodes the intermediate search results. The authors presented a P2C configuration in which the fields are searched using the balanced routing table search (BARTS) scheme [27] in SRAM, and the multidimensional search is implemented using a TCAM. After constructing the table of ternary match strings for each field, the ternary strings associated with each rule are concatenated and stored in TCAM in order of rule priority. A data structure is constructed for each rule field to return the intermediate bit vector for the elementary interval covering the given packet field. These data structures operate in parallel to generate the search key which will be used to query the TCAM.

LECAM [1] is an algorithmic-architectural solution that aims to have the deterministic performance of CAM by blending DCFL with a modified CAM architecture. Similar to DCFL, LECAM uses independent search engines for each rule field, where the search engines are optimized for the type of match condition. LECAM uses the label encoding technique used in DCFL to exploit the redundancy observed in real rule sets to more efficiently represent the set of stored rules. In DECL, the results from the search engines are aggregated in distributed fashion. In LECAM, the search engine results are aggregated using a modified CAM architecture.

Until the time of writing this paper, the state-of-the-art in packet classification is LECAM, which mixes the DCFL packet classification algorithm with a modified CAM. LECAM is an algorithmic-architectural solution that utilizes disintegration and which exploits the observation about real rule sets that the number of unique values in each rule field is much less than the number of rules. LECAM performance is dependent on the high redundancy of unique values in rules fields, and that result in poor performance for LECAM if that property does not hold.

However, the number of rules remains much smaller than the classification space (even if it reaches millions of rules), raising the need for packet classification solutions that utilize the scarcity in the number of rules compared to the classification space. Thus, algorithmic-architectural solutions that are based on cutting the classification space into smaller spaces are expected to be suitable for next generation networks.

3. PACKET CLASSIFICATION FROM A GEOMETRICAL POINT OF VIEW

The packet fields most commonly used in IPv4 packet classification are the 8-bit protocol, 32-bit source address, 32-bit destination address, 16-bit source port, and 16-bit destination port. The "classifier" or "rule database" in a router consists of a finite set of rules, $R1, R2, \ldots, Rn$. Each rule is a combination of k values, one for each header field in the packet. A packet P matches rule Ri if all the packet header fields Pj, $j = 1, \ldots, k$ match the corresponding fields in Ri.

There are three kinds of match: exact match, prefix match, or range match. In an exact match, the header field of the packet must exactly match the rule field. In a prefix match, the rule field is a prefix of the header field, and only that prefix of the header field needs to exactly match the rule field. In a range match, the header values must lie in the range specified by the rule. Range matching is used for specifying port number ranges. Exact match, and prefix match, could be looked at as a special case of range match. For example, the prefix 11** could be represented as a range 12–15, and the exact value 1100 could be represented as the range 12–12.

According to the above definition, a rule can be considered as a hyperplane in k-dimensional space. Rules in the classifier may overlap; consequently, the classifier is a set of overlapping hyperplanes. A packet is a point in the k-dimensional space. Thus, the packet classification problem is equivalent to finding the set of the overlapping hyperplanes that contain the point to be located. This is similar to the point locating problem in computational geometry [28].

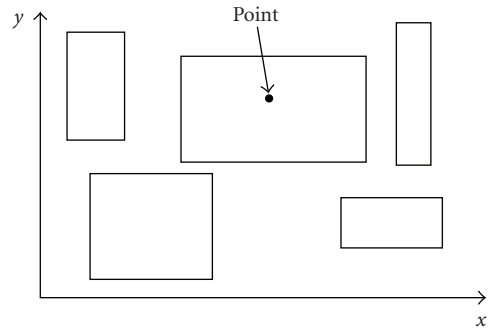

Figure 1: Example for 2-dimensional point locating problem.

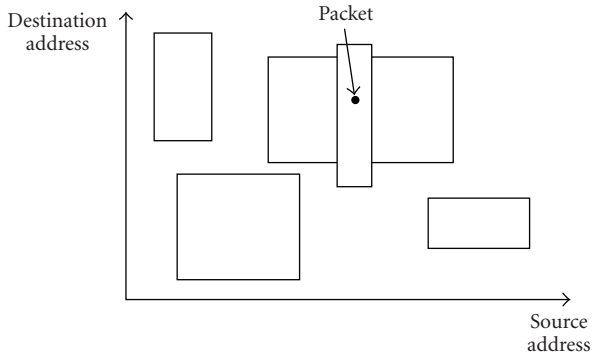

Figure 2: Packet classification is similar to point locating problem in computational geometry but hyperplanes may overlap in packet classification.

The difference between packet classification and point locating problem is in the point locating problem, hyperplanes do not overlap, while in packet classification, hyperplanes may overlap. This makes the packet classification problem more complex than the point locating problem. Nonetheless, structures and characteristics of rules in the classifier may be exploited to reduce the complexity of the packet classification problem in order to get high-performance packet classification algorithms.

The concept of cutting [29, 30] came from computation geometry, where "divide and conquer" principle is applied to reduce the complexity of searching in the classification space to searching in smaller subspaces. The fact that the number of rules in a classifier is much smaller than the classification space means that most of classification space is unoccupied; thus, cutting is a suitable technique to exploit the scarcity of occupied regions in the classification space.

Figures 1 and 2 depict the analog between the packet classification problem and the point locating problem in computational geometry. The example shows that 2-dimensional classification is similar to finding a point in a two-dimensional space. However, in packet classification, a point could be covered by more than one hyperplane. The example of a two-dimensional classification space could be generalized to 5-tuple packet classification.

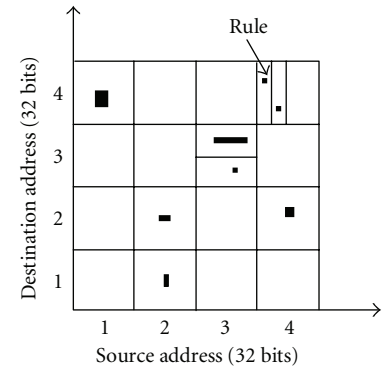

Figure 3: Cutting concept in computational geometry (2-dimensional classifier).

Figure 3 shows a two-dimensional classifier that has 8 rules. The solid black boxes in the figure represent rules. The classification space is cut into subspaces, where each subspace contains only 1 rule or nothing, that is, a bucket size of 1. The bucket size is the maximum number of rules a subspace can contain with no need to divide it further.

At the first level, the classification space is divided into 4 cuts along the source address dimension and 4 cuts along the destination address dimension. That will result in 16 subspaces (cuts). As shown in Figure 3, all of the rectangles have 1 rule or less except the rectangle that lies on the intersection of row 3 and column 3 is denoted as (3,3), and the rectangle that lies on the intersection of row 4 and column 4 is denoted as (4,4). Thus, these two rectangles require further dividing. (3,3) is divided into two smaller rectangles along the destination address dimension. Those two rectangles contain 1 rule each, so no further dividing is required. (4,4) is divided into two subrectangles along the source address dimension. As the left hand one contains two rules, it is divided further into two subrectangles in which each subrectangle contains only 1 rule.

The classification space depicted in Figure 3 represents a multilevel tree as shown in Figure 4. In Figure 4, the number in the boxes represents the number of rules that lie in that subspace (cut). The boxes that have bold frames are nodes to be divided further. The rest of the boxes are leaf nodes. Leaf nodes contain a number of rules less than or equal to a predefined threshold (bucket size) which is 1 in this example.

4. AN ARCHITECTURE FOR CLASSIFICATION BY MULTILEVEL CUTTING

In this section, an architecture that fulfils the following requirements is proposed.

(i) The number of rules in the classification space is much smaller than the classification space, so packet classification by multilevel cutting of the classification space is an appropriate solution.

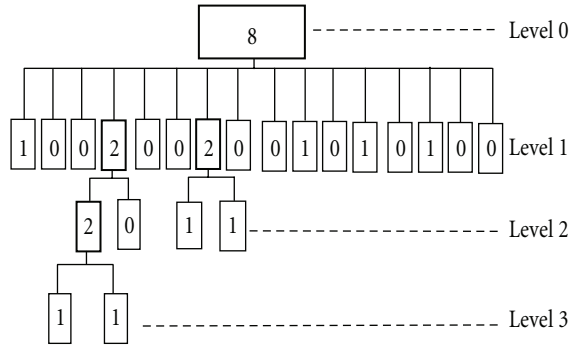

Figure 4: Hierarchy of cutting that represents the classification space in Figure 3.

(ii) The nondeterministic performance of algorithmic solutions makes them undesirable for technology vendors. It is very important for any proposed architecture to have deterministic performance.

(iii) It has been concluded from the state-of-the-art published work [1, 9] that HyperCuts is one of the most promising algorithmic solutions. An algorithmic-architectural solution that mixes HyperCuts with CAMs should provide a good solution which avoids the disadvantages of CAMs and the disadvantages of HyperCuts.

(a) Dynamic updates support is very important for next generation networks.

(b) The advance in technologies, such as ASIC and FPGAs, makes them a desirable solution for system designers for network processing.

4.1. The proposed architecture

For all the above reasons, the architecture shown in Figure 5 is proposed. This architecture gives the required flexibility in trading off the different components of the architecture; it utilizes multidimensional cutting of the classification space and it gives the deterministic performance of CAMs. The proposed architecture is controlled by a microprocessor. The flow chart shown in Figure 6 describes how an incoming packet is classified.

An incoming packet will traverse a decision tree until a matching rule is found. The decision tree is composed of nodes, where branching decisions are made. If the number of rules in a node is less than a predefined threshold (bucket size), then this node is called a leaf node, and it will not be divided further. The incoming packet will be compared with the rules stored in the leaf node to find the matching rule. The number of divisions (cuts) and the dimensions to cut along for each node are decided according to the heuristics used for cutting.

The architecture proposed in Figure 5 is composed mainly of 3 sets of tables: the nodes table, the leaf nodes table(s), and the rules table(s). In addition to the tables, the architecture contains control logic, initial shifts register, comparator(s), CAM, and a priority encoder.

The nodes table consists of a number of entries equal to the number of nodes (including the leaf nodes) in the tree. Each entry is composed of 3 fields: *address*, *type*, and *shifts*. The address field is a pointer to the location of the first node in the subnodes of the current node, if the node is not a leaf node. If it is a leaf node, then the address field is a pointer to the location of the corresponding leaf node in the leaf nodes table. *Type* field determines if the node pointed to is a leaf node or not. Its value is either 1 or 0 (1 bit). *Shifts* field is used when the node is not a leaf node. The width of this field is 18 bits. The shifts field is actually composed of 4 subfields: shifts for the source address (5 bits), shifts for the destination address (5 bits), shifts for the source port (4 bits), and shifts for the destination port (4 bits). The value of the shifts field determines the number of cuts, that is, a value of 3 for the source address shifts means 8 cuts along the source address dimension.

The leaf nodes table(s) contains pointers to the locations of rules stored in the rule tables. The leaf nodes table(s) is/are triggered by the control logic with a signal of width $w7$ which carries the address of the rule in the leaf node. The width of the entries in the leaf nodes table(s) is $\log_2$ (number of rules). The width $w7$ is $\log_2$ (number of leaf nodes $*$ bucket size/number of leaf nodes table(s)). The purpose of having multiple leaf nodes tables is to speed up comparing rules with incoming packets.

For example, instead of having a leaf nodes table that contains 10 leaf nodes with a bucket size of 8, the leaf nodes table could be split into 2 leaf nodes tables each containing 10 leaf nodes with a bucket size of 4, or 4 leaf nodes tables each containing 10 leaf nodes with a bucket size of 2. That will speed up the comparison process to 2 rules at a time or four rules at a time, respectively. The cost for that will be more comparator and rules tables as each leaf nodes table has a corresponding rules table and comparator, provided that the technology used supports such parallelism.

The rules are stored in the rules table(s). The number of entries in this table is the number of rules. Each rule consists of 32 bits for the source address, destination address, source address mask, and destination address mask. In addition to that, it consists of 16 bits for the source port lower value, 16 bits for the source port upper value, 16 bits for the destination port lower value, and 16 bits for the destination port upper value. Thus, each rule requires 192 bits. The number of rule tables is equal to the number of leaf nodes tables. The output of the rules table is the rule and the rule location which is the ID of the rule, so the output is of width $w9$ which is 192 + $\log_2$ (number of rules).

The function of the logic unit is to generate a pointer of width $w2$ to the matching node in the nodes table or to generate a pointer of width $w7$ to the leaf nodes table. $w2$ is equal to $\log_2$ (number of nodes including leaf nodes). $w7$ is equal to $\log_2$ ((number of leaf nodes $*$ bucket size)/number of leaf nodes tables).

The input to the control logic is composed of the incoming packets headers, the initial shifts register, and a signal from the nodes table. The packet header consists of a 32-bit source address, a 32-bit destination address, a 16-bit

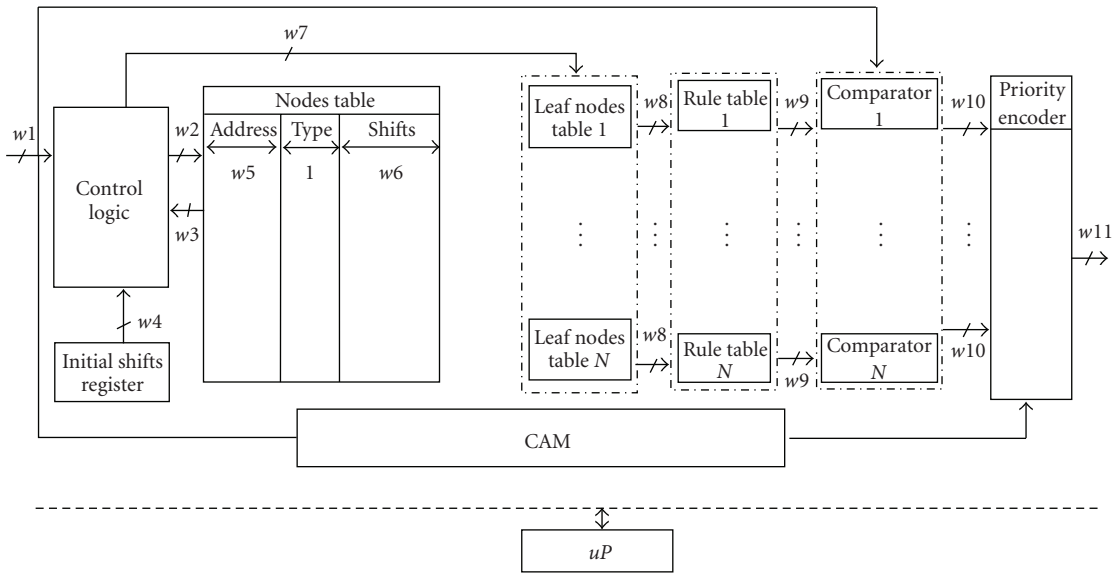

Figure 5: An architecture for multilevel cutting.

source port, a 16-bit destination port, and an 8-bit protocol. Thus, the width $w1$ of packet header fields is 104 bits for the 5-tuple packet classification. In implementations, usually, the protocol field is ignored as it has very few values like TCP, UDP, and ICMP.

The control logic block gets a signal of width $w3$ from the nodes table composed of address, type, and shifts. The address field is a pointer to the location of the first node of the subnodes of the current node. The initial shifts for the highest level are stored in the *initial shifts register* (the location of the first node of the subnodes of the highest level is zero). The width $w3$ is equal to $18 + 1 + \log_2$ (number of nodes including leaf nodes). The control logic uses the incoming packet header with the address and shifts values to point to the *current node location*. If the current node is a leaf node, then the control logic triggers searching the leaf nodes table, otherwise the first node address will be set equal to the value read. The control logic keeps iterating until the node pointed to is a leaf node. Each iteration is equivalent to going one level down the tree.

CAM could be used as a classifier on its own; however, the silicon cost and high-power dissipation make it an undesirable solution. In this architecture, CAM provides a mechanism to guarantee the deterministic performance by controlling the depth of the decision tree (number of levels). By controlling the number of levels, the maximum number of access times required to reach the leaf node is controlled. If the architecture is pipelined, then the maximum number of levels determines the number of pipeline stages. Moreover, CAM increases the performance of cutting by storing rules that cause high redundancy. Another advantage of CAM is supporting dynamic update which is very important for the next generation networks as rules will be set dynamically by the traffic itself. The output of the CAM or the comparators has width $w10 = \log_2$ (number of rules) which is the ID of the rule that will be fed to the priority encoder to resolve any conflict if there is more than one matching rule for a given packet header.

In the following sections, the tradeoffs between the components of the architecture will be analyzed. The parameters are as follows: bucket size, leaf nodes table size, nodes table size, and CAM size.

5. PERFORMANCE ANALYSIS FOR THE PROPOSED ARCHITECTURE

In this analysis, the tradeoffs between the parameters of the architecture are discussed. Testing packet classification solutions remains a difficult task due to the obstacles in obtaining real rule sets and the lack of standard metrics in evaluating packet classification solutions. In order to facilitate packet classification solutions testing, the packet classification benchmarking tool presented in [31] was proposed. This packet classification benchmarking tool was based on 12 real rule sets, the author obtained from communications companies and other researchers. The packet classification benchmarking tool gives flexibility in generating any number of rules based on these rule sets. For confidentiality purposes, the real rule sets were hidden, and the seed files that represent the real rule sets were provided. The number of rules in the real rule sets used to build the packet classification benchmarking tool ranges from 68 to 4557.

The flexibility, given by the packet classification benchmarking tool in generating an infinite number of rule sets based on the 12 seed files, created difficulty in taking a decision about the seed file that should be used, and the number of rules to be generated. According to [3, 9], real rule sets contain less than 5000 rules. Based on that, the number of rules targeted in our analysis is around 5000. In order to

For each dimension
Get the number of unique values
Get the average of the number of unique values over all dimensions
Select the dimensions where the number of unique values ≥ average
Get the number of cuts for each selected dimension

Algorithm 1: Heuristic to select the dimensions to cut along.

generate a synthetic rule set that is close to the real rule set, the seed file chosen is the one that was based on the real rule set that contains 4557 rules, that is, ACL5 in [31]. The parameters for controlling the smoothness and scope of the benchmarking tool are set to zero to keep the synthetic rule set as close as possible to the real one. The benchmarking tool usually generates fewer than the number of rules specified in the command line that is less than the specified number in the command line due to the replication of some generated rules. The number specified in the command line was, therefore, set to 10000, which resulted in generating 5115 rules. It is believed that the number of rules will remain less than 5000 because, in practice, rules are created manually by a system administrator using a standard management tool such as CiscoWorks VPN/security management solution (VMS) [32] or Lucent security management server (LSMS) [33].

The flexibility in the parameters of the heuristics and the components of the architecture gives enough flexibility in choosing the most suitable values for a given rule set. There are two heuristics used in building the multilevel decision tree. The first one is to select the dimensions to cut along, and the second one is to select the number of cuts per dimension. These two heuristics were based on what is published in [3]. In our implementation, the heuristic shown in Algorithm 1 was used to select the dimensions to cut along. The heuristic selects the dimensions, where the number of unique values is larger than the average number of unique values in all dimensions.

For each selected dimension, heuristic shown in Algorithm 2 is used to select the number of cuts.

This heuristic keeps doubling the number of cuts until the percentage of empty nodes exceeds a predefined percentage. The number of total cuts at each node is limited to a predefined maximum. This number is limited in order to avoid an explosive growth in the number of nodes.

The above two heuristics will be used at each node until the number of rules falls below the predefined bucket size. The focus of this analysis is the tradeoffs between the components of the architecture rather than the parameters of the algorithms themselves. Thus, the maximum number of cuts is kept fixed at 128, and the maximum percentage of empty nodes is kept fixed at 0.9 throughout the analysis. It is believed that 128 and 0.9 will give flexibility for the heuristics to keep cutting along a selected dimension, so that most of the resulting subrule sets will have a number of rules less than the bucket size.

The analysis was carried out for bucket sizes 8, 16, 32, 64, 128, and 256. The results targeted are the number of nodes, the number of leaf nodes, and the maximum number of levels. As this architecture must achieve the deterministic performance of CAMs, the number of levels measured is the maximum value (worst case) rather than the average value.

Figures 7, 8, and 9 depict the effect of increasing bucket size on the number of levels, number of nodes, and number of leaf nodes. The number of levels represents the number of times the control logic has to access the nodes table to get the location of the leaf node or the number of pipeline stages if the architecture is pipelined. Number of nodes represents the size of the nodes table. Number of leaf nodes multiplied by the bucket size represents the size of the leaf nodes table.

These graphs are useful due to the fact that for a given rule set, the system designer is able to decide what the best bucket size is for the technology that will be used in implementation according to the limitations of the technology, and the parameters of the architecture such as speed and power consumption. In order to illustrate this, two examples are given below.

If the bucket size is 8, then the number of levels is 14, the number of nodes is 5.5 millions, and the number of leaf nodes is 2500. To store 5.5 million nodes, off-chip memory is required. The control logic requires accessing the memory 14 times to reach the leaf node (assuming that 1 access to the memory is enough to read the required data about the node). Off-chip QDR SRAM works at 300 MHz (word length of 32 bits) [34] which means that 1 access to memory requires 3.3 nanoseconds; which implies that by using off-chip SRAM that works at a speed of 300 MHz, the control logic requires 46.2 nanoseconds to reach the leaf node which implies that the classifier cannot classify more than 21.6 million packets per second (calculations are based on the worst case scenario in order to guarantee the deterministic performance).

The other extreme of the bucket size is 256, where the number of levels is 5, the number of nodes is 7625, and the number of leaf nodes is 180. Assuming on-chip memory that is operating at 550 MHz nanoseconds [35], and an implementation technology that supports enough parallelism in searching a leaf node of size 256 for 180 leaf nodes to find the matching rule within the 5 times required to access the nodes table. Under that assumption, 5 accesses to memory require 10 nanoseconds, which means that the classifier speed cannot exceed 100 million packets per second. For a pipelined implementation, the classifier speed will rise to 500 million packets per second. This number is very promising provided that finding the matching rule in the leaf node does not require more than 1 access to the memory. Due to the technology limitations, in real implementation, finding the matching rule in the leaf node requires multiple accesses to the memory. This means that the speed of the classifier will drop by a factor equal to the required number of accesses to the memory.

```
Initialize max No. of cuts to a predefined value
Initialize max % of empty nodes to a predefined value
No. of cuts = 1
Loop
    No. of empty nodes = 0
    Loop on cuts
        Get the range to search in
        No. of rules in this cut = 0
        Loop on rules
            If rule is in the range then increment the No. of rules in this cut
            If No. of rules is this cut is 0 then increment the No. of empty nodes
    Calculate the % of empty nodes to total nodes
    If (% > max % or empty nodes \\ number of cuts ≥ max number of cuts)
        break
    else
        double the number of cuts
```

Algorithm 2: Get number of cuts for each selected dimension.

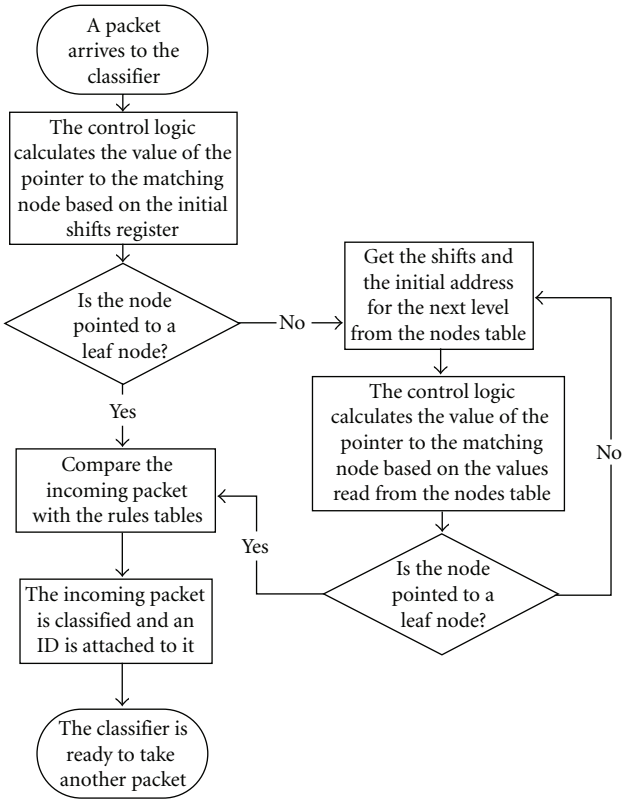

Figure 6: A flow chart for classifying the incoming packets.

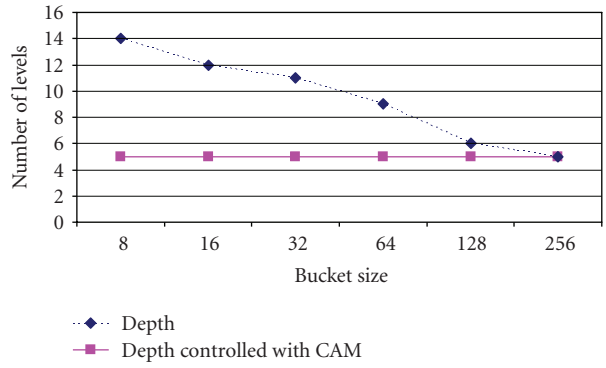

Figure 7: Restricting maximum depth of the tree (number of levels) to 5 using CAM.

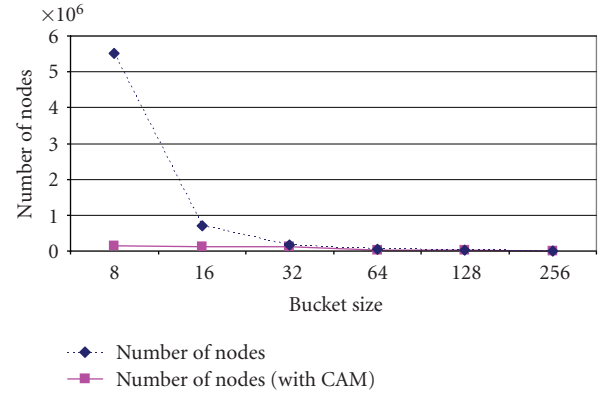

Figure 8: The effect on the number of nodes of restricting the maximum depth of the tree (number of levels) to 5 using CAM.

Increasing the bucket size comes at the cost of larger memory requirements. As shown in Figure 9, increasing the bucket size from 8 to 256 resulted in decreasing the number of leaf nodes from 2501 to 180, but the overall entries in memory = the bucket size $*$ number of leaf nodes. This implies that by increasing the bucket size from 8 to 256, the memory requirements increased from 20008 to 46080 entries. Each entry is composed of 104 bits for the 5 fields, and 104 bits for the mask (192 bits if the protocol field is ignored).

Moreover, increasing the bucket size could result in decreasing the speed of the classifier. This is due to the

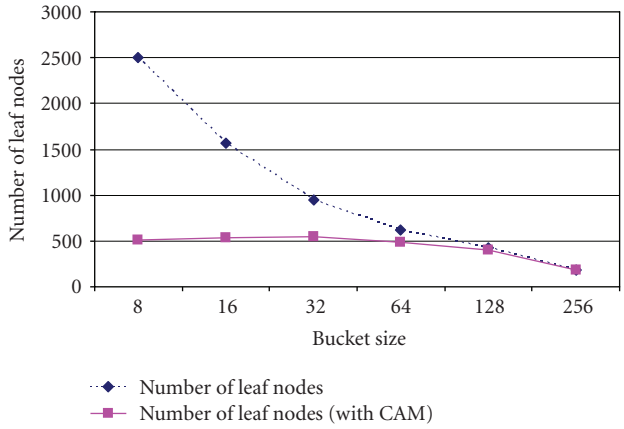

Figure 9: The effect on the number of leaf nodes of restricting the maximum depth of the tree (number of levels) to 5 using CAM.

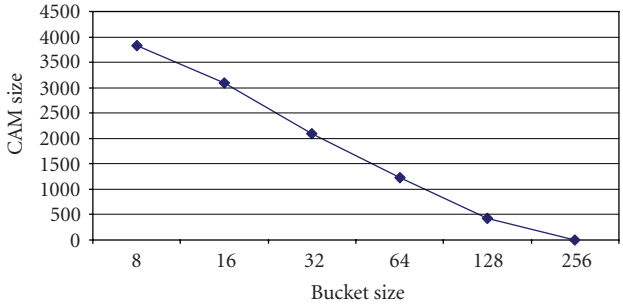

Figure 10: CAM size required to accommodate rules moved to CAM.

fact that increasing the bucket size increases the number of accesses to the memory to find the matching rule.

The system designer should optimize the architecture according to the limitations of the technology used in the implementation, and the preferences of the design.

5.1. Controlling the maximum number of levels using CAM

The previous results show that the maximum number of levels decreases as the bucket size increases. However, the bucket size cannot guarantee that the maximum number of levels will not exceed a certain limit. This could be guaranteed by using CAM as the tree will be trimmed at a certain depth, and rules in the trimmed nodes will be moved to the CAM.

In the following figures, the bucket size will be variable, and the number of levels will be kept constant at 5. Figures 7, 8, and 9 show the effect of using CAM to force the number of levels to be 5. Any number of levels other than 5 could be used, as the results should follow the same trends.

In general, the number of nodes and the number of leaf nodes decrease by using CAM. The effect of using CAM is more significant when the bucket size is small. However, this comes at the cost of using larger CAM as shown in Figures 10 and 11. Controlling the number of levels guarantees

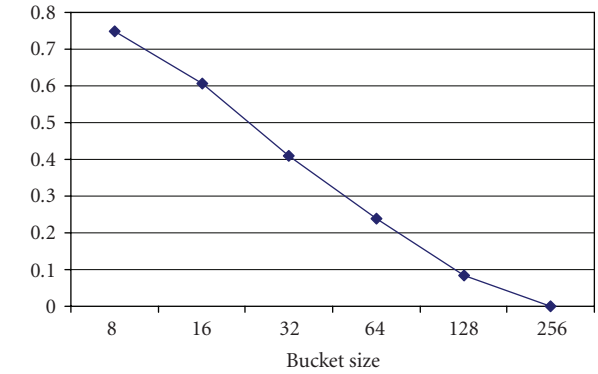

Figure 11: CAM size required to accommodate rules moved to CAM as percentage to the total number of rules.

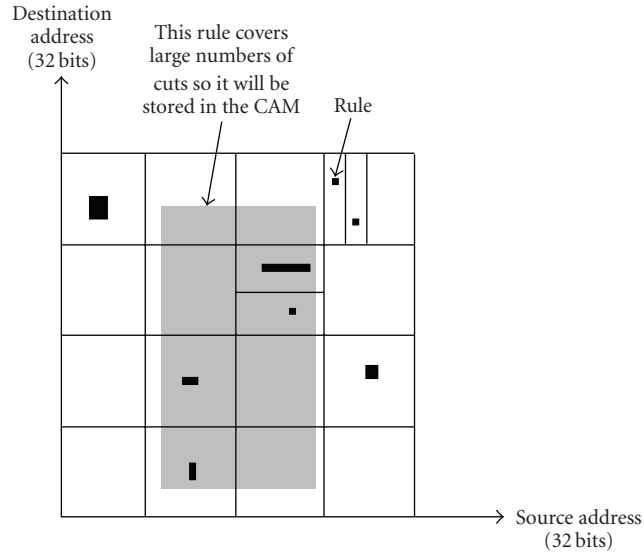

Figure 12: Some rules are redundant in many cuts.

controlling the maximum number of pipelining stages or controlling the worst case delay that is caused by accessing the nodes table. The effect of using CAM to control the maximum number of levels on the number of nodes and the number of leaf nodes is more significant for the small bucket sizes like 8 and 16. This is due to the fact that the tree will be trimmed from 14 levels to 5, and rules will be stored in CAM instead of the leaf nodes. The corresponding CAM sizes for bucket size 8 and 16 are 75% and 60%, respectively, which means that the architecture could be looked at as a CAM with increased capacity or reduced power consumption. The CAM size approaches zero as the bucket size increases due to the fact that the number of levels decreases as the bucket size increases. This implies that the silicon cost of the architecture is mainly in the RAMs as CAM size approaches zero.

5.2. Tackling redundant rules using CAM

Some of the rules in the classifier might be redundant in large numbers of cuts as shown in Figure 12. The redundant rules

Loop rules
Width (w1) of source address = (32 − mask width)
Width (w2) of destination address = (32 − mask width)
Width (w3) of source port = (upper limit − lower limit) rounded to the closest number of bits
Width (w4) of destination port = (upper limit − lower limit) rounded to the closest number of bits
Overall width = w1 + w2 + w3 + w4
Record the overall width of the current rule in ordered list
Move a predefined number of the largest width hyper-planes (rule) to the CAM
Build the tree

Algorithm 3

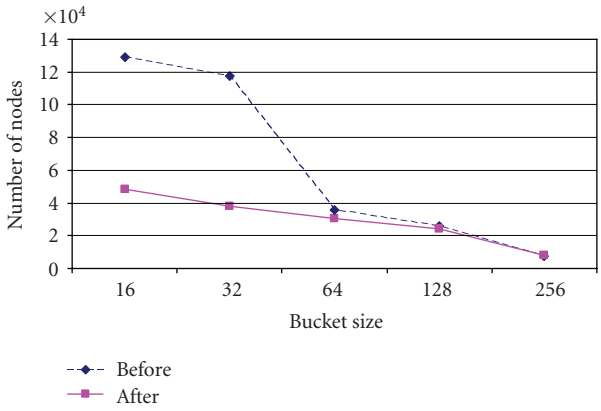

Figure 13: The effect on the number of nodes of moving the largest 10% of the rules.

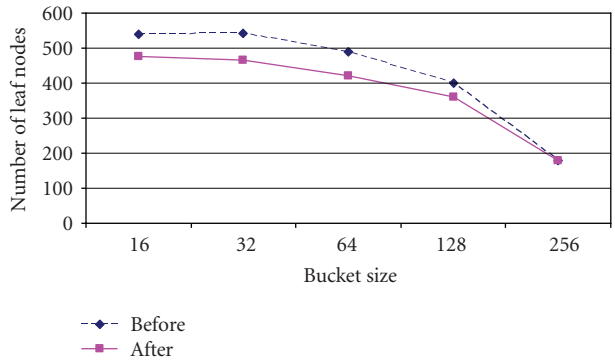

Figure 14: The effect on the number of leaf nodes of moving the largest 10% of the rules.

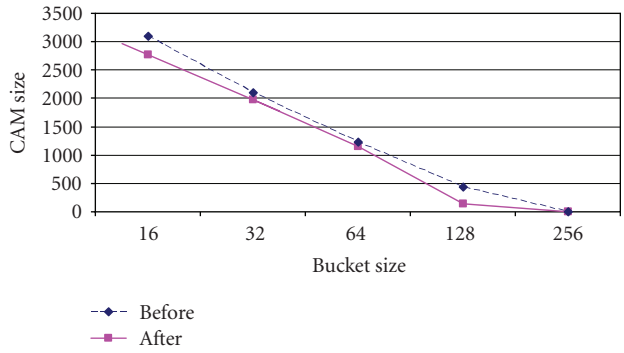

Figure 15: The effect on the CAM size of moving the largest 10% of the rules.

could be tackled by storing them in the CAM before building the tree.

Heuristic shown in Algorithm 3 is used to select the largest rules (hyperplanes) that cover the largest areas of the classification space, and then moves them to the CAM.

Figures 13, 14, and 15 show the effect of removing the largest rules (hyperplanes) on the number of nodes, number of leaf nodes, and the size of the CAM used to control the maximum number of levels (depth of the decision tree) to 5. In this example, the top 10% of the rules (sorted by the largest) will be moved to CAM before building the tree. As shown in Figures 13 and 14, there is a reduction in the number of nodes and in the number of leaf nodes. The percentage of reduction goes down as the bucket size goes up.

The effect of moving the largest 10% of the rules to CAM reduces the rules set size from 5115 to (5115–510), but it adds 510 rules to the CAM. Figure 15 shows that there is a reduction in the CAM size; however, moving the 10% rules to CAM, and this increases the CAM size. The effect of that on the number of nodes and number of leaf nodes is a significant reduction especially for the small bucket sizes. For example, for a bucket size of 16, the number of nodes went down from 130000 to 50000—a reduction of 60%. That result in a saving of memory especially if on-chip memory is used. The same thing applies for leaf nodes, where there is a reduction from 550 to 450 leaf nodes, which results in a saving in the resources used. The saving in resources comes at the cost of an increase in the CAM size. The CAM size went down by 350 from 3100 to 2750. There is not actually a reduction as 510 rules should be added to the CAM; thus, CAM size effectively increases by 160. So, there is a tradeoff between the increase in the CAM size and the decrease in number of nodes and number of leaf nodes.

The effect of moving rules that cover large areas keeps decreasing as the bucket size increases. The reason is that rules that cover large areas cause the cutting heuristic to increase the number of cuts which is and that results large number of nodes.

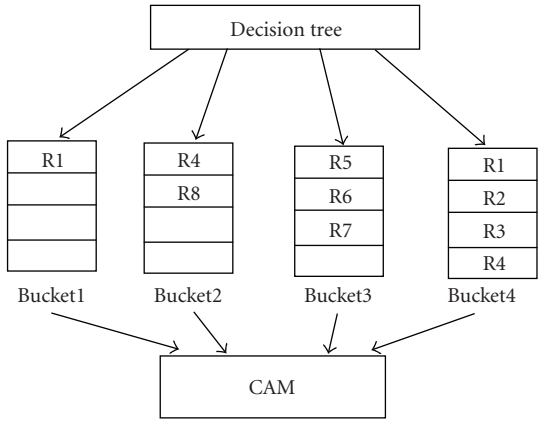

Figure 16: An example to illustrate dynamic updates: rules that cannot be inserted in buckets will be inserted in CAM.

6. DYNAMIC UPDATES

The classifier supports dynamic updates if rules are inserted and deleted without the need to rebuild the decision tree. To explain dynamic updates for packet classification by multilevel cutting of the classification space, consider the decision tree depicted in Figure 16. This classifier has a bucket size of 4 with rules distributed on buckets as shown in Figure 16.

The classifier shown in Figure 16 has 6 empty locations in the buckets, so the maximum number of rules that can be inserted in the classifier without rebuilding the decision tree is 6. However, if a rule is going to be inserted in Bucket 4, then the tree requires rebuilding or that rule cannot be inserted in the classifier. This means that the classifier can support inserting any number of rules between 0 and 6 depending on where the rules are going to be inserted. So, without CAM, the classifier can support low to average rate of dynamic updates. The classifier can support dynamic updates until a condition happens, where the number of rules to be inserted in a bucket is more than the number of empty locations in that bucket such as the following.

(i) A rule is required to be inserted in Bucket 4.

(ii) Four rules are required to be inserted in Bucket 1,

or any other conditions that result in overflowing the buckets.

Thus, the probability that the tree requires rebuilding is the probability that an insertion of a rule results in an overflow in one of the buckets. The highest probability is the probability that a rule is inserted in Bucket 4.

Assuming a model in which the rules inserted are randomly distributed over the buckets, then the probability that a rule is inserted in Bucket 4 is 1/4. The probability that a second rule is inserted in the same bucket is (1/4) $*$ (1/4) which is 1/16, and so on. If N rules are supported by CAM, then the probability is $1/(4 \wedge (\text{CAM size}))$ or generally $1/((\text{number of buckets}) \wedge (\text{CAM size}))$. Obviously, this number approaches zero exponentially as the CAM size increase linearly.

Moreover, dynamic updates are not only insertions for rules as there are rules going to be deleted. For a classifier that supports certain number of rules, it is expected in average that for each rule inserted, there will be a rule deleted. This makes the need for rebuilding the tree is even less probable. If there is any transient case in which the number of rules inserted in higher that the number of rules deleted then CAM will accommodate this transient condition.

The proposed architecture has the capability to support a high rate of dynamic updates. However, the decision tree could be rebuilt optionally to optimize the distribution of rules over cuts.

7. CONCLUSION

A new architecture for packet classification is proposed based on multilevel cutting of the classification space into subspaces. The proposed architecture takes advantage of the HyperCuts packet classification algorithm, the geometrical distribution of rules in the classification space, and CAMs. The proposed architecture gives the deterministic performance of CAMs with flexibility for system designers to trade off the components of the architecture according to the technology limitations and to the preferences of the system (speed, power dissipation). The proposed architecture was analyzed for the different tradeoffs between the bucket size, number of nodes, number of leaf nodes, and number of levels. CAM was used in the architecture to limit the number of levels, to remove redundant rules, and to support dynamic updates.

Washington University in St. Louis proposed DCFL, the best algorithmic solution in the "disintegration" category, and based on DCFL, they proposed ELCAM which is the state-of-the-art in algorithmic-architectural solutions. DCFL and ELCAM are based on the observation that the number of unique values in each rule field is very small compared to the number of rules. The number of unique values in the rule fields has significant impact on the performance of DCFL and ELCAM. If the number of unique values in rule fields increases, then the performance of DCFL and ELCAM declines. Dynamic updates support for DCFL and ELCAM is dependent on the number of unique values in rule fields. For the next generation networks where the number of rules reaches millions, and rules are dynamic and set by the traffic itself, it is not expected that the number of unique values in each rule field will remain very small compared to the number of rules.

The architecture proposed in this paper is able to support next generation networks as it is based on the fact that the number of rules is small compared to the classification space. The characteristics of the rules could be exploited to make the cutting more optimal. Moreover, ELCAM does not provide the flexibility provided in this architecture for the system designer to trade off the components of the architecture according to the limitations of the technology used and the preferences of the system designer. The existence of the aggregation CAM in ELCAM, which has a size dependent on the characteristics of the rules, restricts the

ability of the system designer to trade off the components of the architecture.

The proposed architecture takes HyperCuts, the-state-of-the-art packet classification algorithm in the "cutting and tries" category, from theory to implementation on emerging technologies. Technology providers are reluctant to support algorithmic solutions including HyperCuts due to their nondeterministic performance. The proposed architecture supports dynamic updates at high rates which were not supported in HyperCuts, and it provides the deterministic performance of CAMs.

REFERENCES

[1] D. E. Taylor and E. W. Spitznagel, "On using content addressable memory for packet classification," Tech. Rep. WUCSE-2005-9, Department of Computer Science & Engineering, Washington University, Saint Louis, Mo, USA, September 2005.

[2] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," in *Proceedings of the 7th IEEE Hot Interconnects Symposium (HotI '99)*, pp. 34–41, Palo Alto, Calif, USA, August 1999.

[3] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '03)*, pp. 213–224, Karlsruhe, Germany, August 2003.

[4] T. Y. C. Woo, "A modular approach to packet classification: algorithms and results," in *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '00)*, vol. 3, pp. 1213–1222, Tel Aviv, Israel, March 2000.

[5] D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner, "Router plugins: a software architecture for next generation routers," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '98)*, pp. 229–240, Vancouver, Canada, August-September 1998.

[6] A. Feldmann and S. Muthukrishnan, "Tradeoffs for packet classification," in *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '00)*, vol. 3, pp. 1193–1202, Tel Aviv, Israel, March 2000.

[7] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '98)*, pp. 191–202, Vancouver, Canada, June 1998.

[8] P. Gupta and N. McKeown, "Packet classification on multiple fields," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '99)*, pp. 147–160, Cambridge, Mass, USA, August 1999.

[9] D. E. Taylor and J. S. Turner, "Scalable packet classification using distributed crossproducing of field labels," in *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '05)*, vol. 1, pp. 269–280, Miami, Fla, USA, March 2005.

[10] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '98)*, pp. 203–214, Vancouver, Canada, August 1998.

[11] F. Baboescu and G. Varghese, "Scalable packet classification," *IEEE/ACM Transactions on Networking*, vol. 13, no. 1, pp. 2–14, 2005.

[12] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '99)*, pp. 135–146, Cambridge, Mass, USA, August 1999.

[13] V. Srinivasan, "A packet classification and filter management system," in *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '01)*, vol. 3, pp. 1464–1473, Anchorage, Alaska, USA, April 2001.

[14] L. Chisvin and R. J. Duckworth, "Content-addressable and associative memory: alternatives to the ubiquitous RAM," *Computer*, vol. 22, no. 7, pp. 51–64, 1989.

[15] K. E. Grosspietsch, "Associative processors and memories: a survey," *IEEE Micro*, vol. 12, no. 3, pp. 12–19, 1992.

[16] T. Kohonen, *Content-Addressable Memories*, Springer, New York, NY, USA, 2nd edition, 1987.

[17] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: a tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[18] S. Stas, "Associative processing with CAMs," in *Proceedings of the IEEE Northwest Conference (Northcon '93)*, pp. 161–167, Portland, Ore, USA, October 1993.

[19] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '97)*, pp. 25–36, Cannes, France, September 1997.

[20] R. A. Kempke and A. J. Mcauley, "Ternary CAM memory architecture and methodology," US patent no. 5841874, Motorola, Inc., Schaumburg, Ill, USA, November 1998.

[21] G. Gibson, F. Shafai, and J. Podaima, "Content addressable memory storage device," US patent no. 6044005, SiberCore Technologies, Inc., Ontario, Calif, USA, March 2000.

[22] Datasheet, "Harmony TCAM 1 Mb and 2 Mb," Micron Technology Inc., January 2003.

[23] Datasheet, "36Mb DDR SIO SRAM 2-Word Burst," Micron Technology Inc., December 2002.

[24] F. Yu and R. H. Katz, "Efficient multi-match packet classification with TCAM," in *Proceedings of the 12th Annual IEEE Symposium on High Performance Interconnects (Hot Interconnects '04)*, pp. 28–34, Stanford, Calif, USA, August 2004.

[25] K. Zheng, H. Che, Z. Wang, B. Liu, and X. Zhang, "DPPC-RE: TCAM-based distributed parallel packet classification with range encoding," *IEEE Transactions on Computers*, vol. 55, no. 8, pp. 947–961, 2006.

[26] J. van Lunteren and T. Engbersen, "Fast and scalable packet classification," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, pp. 560–571, 2003.

[27] J. van Lunteren, "Searching very large routing tables in wide embedded memory," in *Proceedings of the IEEE Global Telecommunicatins Conference (GLOBECOM '01)*, vol. 3, pp. 1615–1619, San Antonio, Tex, USA, November 2001.

[28] F. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer, New York, NY, USA, 1985.

[29] B. Chazelle, *Discrete and Computational Geometry*, Springer, New York, NY, USA, 1993.

[30] M. H. Overmars and F. A. van der Stappen, "Range searching and point location among fat objects," *Journal of Algorithms*, vol. 21, no. 3, pp. 629–656, 1996.

[31] D. E. Taylor and J. S. Turner, "ClassBench: a packet classification benchmark," *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, pp. 499–511, 2007.

[32] Cisco, "Ciscoworks VPN/security management solution," Tech. Rep., Cisco Systems, Inc., San Jose, Calif, USA, 2004.

[33] Lucent, "Lucent security management server: security, VPN, and QoS management solution," Tech. Rep., Lucent Technologies, Inc., Murray Hill, NJ, USA, 2004.

[34] L. Gopalakrishnan, "QDR II SRAM Interface for Virtex-5 Devices," Xilinx, XAPP853 (v1.1), January 2008.

[35] Data sheet, "Virtex -5 Family Overview," Xilinx, DS-100 (v3.4), December 2007.