Proceedings of the IASTED International Conference
Communication, Network, and Information Security
October 9–11, 2006, MIT, Cambridge, MA, USA
ISBN CD: 0-88986-636-8

TOWARDS EFFECTIVE PACKET CLASSIFICATION

Yaxuan Qi[1] and Jun Li[1, 2]
[1] Research Institute of Information Technology (RIIT), Tsinghua University, Beijing, China, 100084
[2] Tsinghua National Lab for Information Science and Technology (TNLIST), Beijing, China, 100084
{yaxuan, junl}@tsinghua.edu.cn

ABSTRACT
A variety of network security services, such as access control in firewalls and protocol analysis in intrusion detection systems, require the discrimination of packets based on the multiple fields of packet header, which is called Multidimensional Packet Classification. In this paper, we propose a very effective packet classification algorithm called Extended Multidimensional Cuttings, ExCuts in short. As an extension of HyperCuts, which is the best-known existing decision tree algorithm, ExCuts refines the node merging mechanism using a two-step discontiguous space aggregation scheme, which minimizes the number of child nodes. To further reduce the memory usage of the decision tree structure, ExCuts adopts a bit string mapping scheme to compress the large pointer arrays in internal nodes. Due to the significant memory reduction, ExCuts is able to pick a fixed number of cuttings and thus provides explicit worst-case search time. Experimental results show that ExCuts outperforms the best result of existing algorithms on both real-life rulesets and synthetic classifiers.

KEY WORDS
Network security, Packet classification, ACL and IDS

1. Introduction

Multidimensional Packet Classification is crucial to modern network security devices such as firewalls and intrusion detection systems. Although hardware like Ternary CAMs provide multi-Gbps classification rate, they consume too much power and board area and are not cost-effective for applications with large number of rules. To overcome the limits in hardware solutions, there is an increasing interest in both industry and academia in finding efficient software approaches for multidimensional packet classification. The need for extensive study on novel multidimensional packet classification algorithms comes from:

a) Performance of Existing Algorithms: Current algorithms with the best time-space tradeoffs appear to be HiCuts and its extension HyperCuts. Although it is reported in [1] and [7] that these two algorithms work well for real-life rulesets, the performance of HiCuts and HyperCuts is still limited by the following factors:

- *Non-deterministic worst-case search time*: Although worst-case search time is the most important performance metric in packet classification, both HiCuts and HyperCuts do not provide explicit worst-case bounds for it.
- *Excessive memory usage for large rulesets*: A test on the largest real-life ruleset with 1,945 rules shows that even HyperCuts requires more than 1M bytes memory storage, which precludes the use of a common CPU cache.

b) Novel Ideas: Although a number of papers on algorithmic solutions for multidimensional packet classification have been published [1, 3, 4, 5, 6, 7, 8, 9] in recent years, there are still novel ideas that can further improve the performance of existing best algorithms. Novel ideas spring from:

- *New characteristics*: Real-life rulesets have some inherent structure which can be exploited by packet classification algorithms. In recent literature [17], a number of statistic results on real-life rulesets are proposed, providing benefits for algorithmic studies.
- *Hybrid algorithms*: No single algorithm will perform well for all cases. Hence a hybrid scheme might be able to combine the advantages of several different approaches.
- *Efficient search structures*: The search structure built by a lot of algorithms can be significantly compressed. Carefully designed data-structures may also improve the search speed [6].

In this paper, we introduce a novel packet classification algorithm which significantly improves the performance of existing best-known algorithms. Because the proposed algorithm originates from HyperCuts [7], we name our algorithm Extended Multidimensional Cuttings, ExCuts in short. Compare to the HyperCuts, the improvement of ExCuts includes:

a) Guarantees the worst-case search time: Worst-case search time is the most important performance metric in packet classification. However, due to the unpredictable depth of the decision tree built by HyperCuts, it appears to be difficult to provide the worst-case search time. ExCuts guarantees a worst-case search time by fixing the number of partitions along each internal node, but on the same time it remains a modest memory usage.

b) Aggregates discontiguous sub-spaces: Space aggregation is the key step for packet classification algorithms to eliminate the structural redundancy in real-

life rulesets. To maximize the reuse of child nodes, ExCuts aggregates not only the contiguous search spaces (as in HiCuts and HyperCuts), but also the discontiguous search spaces. This scheme significantly reduces the memory storage in comparison with HyperCuts.

c) Compresses redundant pointer arrays: HyperCuts uses pointer array to lead the way for packet search. The use of pointer array can, however, increase storage because the size of these arrays might be large. ExCuts employs an aggregation bit-string mechanism to effectively compress the pointer array without significant loss of search rate.

The rest of the paper is organized as follows: Section 2 states the problem of packet classification; Section 3 and Section 4 describe the proposed algorithm ExCuts; Section 5 illustrates the experimental results; and as a summary, Section 6 shows our conclusions.

2. Problem Definition

Multidimensional packet classification classifies a packet based on F fields of the packet header header. Each rule has F components, and the i^{th} component of rule R, referred to as R[i], is a regular expression on the i^{th} field of the packet header. A packet P is said to match a particular rule R, if $\forall i$, the i^{th} field of the header of P satisfies the regular expression R[i]. It is possible for a packet P to match multiple rules, and in this case, the final classification result is the matching rule with the highest priority in the ruleset.

From a geometric point of view, all possible attributes in F fields of the packet header form a multidimensional space, which is called the *search space*. Each of the F fields is a *dimension* of the search space and a packet P is a *point* located in the multidimensional search space. For range matching in generalized packet classification, the regular expression $R[i]$ refers to a range in the i^{th} dimension of the search space and all ranges specified by rule R compose an F-dimensional hyper-rectangle. If a packet P matches a particular rule R, the point represented by P will fall into the hyper-rectangle specified by R. Therefore, one possible approach in theoretical analysis is to map packet classification into a geometric point location problem in a multidimensional search space.

It has been proved in [2] that the best bounds for point location in N non-overlapping F-dimensional hyper-rectangles are $\Theta(\log^{F-1} N)$ search time with $\Theta(N)$ storage, or $\Theta(\log N)$ search time with $\Theta(N^F)$ storage. In packet classification problem, rules (hyper-rectangles) may overlap, making classification at least as hard as point location. Moreover, the need to match on ranges as well as prefixes makes multidimensional packet classification yet more complex. According to [19], the range-to-prefix conversion for a rule with W-bit range specification in each of the F fields will generate up to $(2(W\text{-}1))^F$ times more rules.

3. Extended Multidimensional Cuttings

Due to the worst-case theoretical bounds found in computational geometry, it seems to be impossible to design a single algorithm that performs well for all cases. Fortunately, real-life rulesets have some inherent characteristics that can be exploited to reduce the complexity both in search time and memory space [19]. In recent literature, a number of multidimensional packet classification algorithms have been proposed [1, 3, 4, 5, 6, 7, 8, 9], some of which achieve promising results by exploiting the structural redundancy found in real-life ruleset. In this paper, the proposed algorithm, Extended-multidimensional Cuttings, improves the performance of the best reported algorithm HyperCuts by introducing the following novel ideas.

3.1 Fixed Number of Cuttings

Worst-case bound of search time is critically important for packet classification algorithms. Unfortunately, both HiCuts and HyperCuts can hardly provide an explicit worst-case search time because the number of cuttings at each level of the decision tree is not a constant. However in ExCuts, the number of cuttings is fixed by setting a constant stride w, i.e. at each internal node the current search space is partitioned into 2^w cuttings along each dimension. This guarantees a worst-case bound of $\Theta(\frac{W}{wf})$, where W is the bit-width of packet header and f is the number of dimensions to cut. By choosing a larger constant stride w, the worst-case search time can be improved.

Although the choice of a larger number of cuttings tends to require more memory storage, ExCuts makes the memory requirement not sensitive to the number of cuttings by effective space compressing mechanism. Experiment results show that when stride w varies from 4 to 7, the total memory space occupied by the search structure varies within the same order of magnitude.

Preprocessing for heuristic algorithms is time-consuming when dealing with large rulesets. To determine a proper number of cuttings for an internal node, HiCuts and HyperCuts will search for an optimized stride *exhaustively* within a set of possible values. While in ExCuts, the complicated optimizations are significantly simplified by using a fixed number of cuttings, and hence the preprocessing time in building the decision tree is greatly reduced.

3.2 Discontiguous Space Aggregation

Space aggregation is the key step to reduce the spatial redundancy in real-life rulesets [19]. Existing decision tree algorithms, such as HiCuts and HyperCuts, only aggregated contiguous sub-space.

Discontiguous space aggregation will generate minimized number of child nodes for each internal node. Compared

to HyperCuts, where child nodes with discontiguous search space will not be aggregated even if they share the same set of rules, ExCuts generates only one child node for each unique ruleset. Consequently, the number of child nodes is significantly reduced and thus the memory space occupied by the decision tree is greatly saved. Although preprocessing of ExCuts to generate a single internal node is slower than HyperCuts (need a search in *f*-dimensions), the overall preprocessing time of ExCuts is much shorter in most of the cases because the total number of nodes is significantly reduced.

3.3 Compressed Point Arrays

To further reduce the memory storage, ExCuts compresses the large pointer arrays in HyperCuts by a two-step mapping:

- In the first step, contiguous cuttings are aggregated along each dimension. ExCuts uses *aggregation bit strings* to compress the pointer array.
- In the next step space aggregation is performed in *f* dimensions. Pointers correspond to aggregated sub-spaces are associated with the child node.

3.3 An Example

For easy understanding of the novel ideas in ExCuts, we provide an example on a toy ruleset shown in Figure 1. Assume that there are 4 cuttings along both X and Y dimensions. The search space is partitioned into 4*4=16 sub-spaces (Figure 2).

In the first aggregation step, contiguous cuttings are aggregated if they share the same set of rules. In Figure 3, Cut1x and Cut2x are aggregated in X dimension, and similarly, Cut1y and Cut2y are combined along Y dimension. Thus the 16 sub-spaces are aggregated to 9 sub-spaces. Figure 4 shows how to map a Cut to the corresponding AC (aggregated cutting) using aggregation bit strings (ABS).

To further cut down the number of sub-spaces, ExCuts aggregates discontiguous sub-spaces by creating a 3*3 pointer array. These pointers (indexed by ACs) are pointed to 5 child nodes, each of which has distinct rules. Figure 5 shows the pointer array build by ExCuts, and for comparison, Figure 6 shows the pointer array build by HyperCuts.

From this example we see that ExCuts significantly reduces the number of child nodes, as well as the size of the pointer array. We will describe details of ExCuts in the next section.

4. The Proposed Algorithm

To hierarchically decompose the multidimensional search space, ExCuts adopts a decision tree structure. At each internal node the current search space along with the corresponding ruleset is split based on information from multiple fields in the rules.

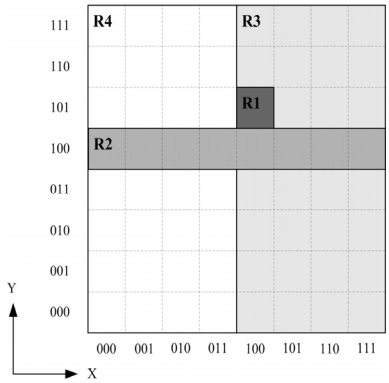

Figure 1. A Toy Ruleset of Four 2-D Rules. The search space is the X-Y plane {[000, 111], [000, 111]}. The ranges specified by rules R1~R4 are: R1{[100, 100], [101, 101]}, R2{[000, 111], [100, 100]}, R3{[100, 111], [000, 111]} and R4{[000, 111], [000, 111]}. R1~R4 are assiged with decremental priorities.

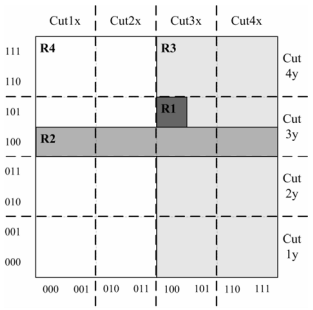

Figure 2. Space Partition. There are 4 cuttings along both X (Cuts1x~Cuts4x) and Y (Cuts1y~Cuts4y) dimensions. The search space {[000, 111], [000, 111]} is partitioned into 4*4=16 equal-sized sub-spaces.

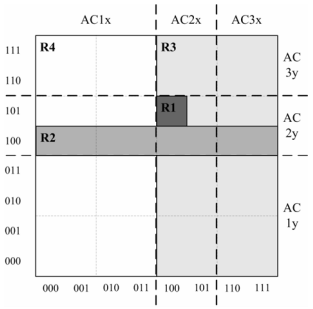

Figure 3. Contiguous Space Aggregation (16-to-9). Cut1x and Cut2x are aggregated in X dimension to form Aggregated Cuttings AC1x. Similarly, Cut1y and Cut2y are aggregated to form AC1y. Now the 3*3 Aggregated Cuttings reduce the number of sub-spaces from 16 to 9.

Cuts	Cuts1x	Cuts2x	Cuts3x	Cuts4x
ABS	1	0	1	1
ACs	AC1x		AC2x	AC3x

Figure 4. Aggregation Bits String. If a packet drops in Cuts3x, then by computing the sum of the first 3 bits of the ABS: 1+0+1=2, we know this packet belongs to AC2x.

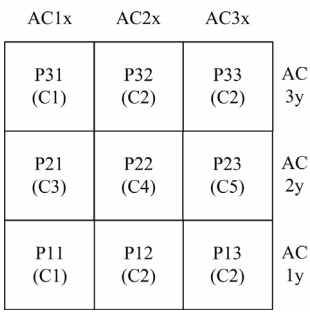

Figure 5. Pointer array Discontiguous Space Aggregation (9-to-5). For example, sub-space (AC1x, AC1y)={[000, 001], [000, 011]} and sub-space (AC1x, AC3y)={[000, 001], [110, 111]} have the same colliding ruleset {R4}, they are aggregated to Child Space C1{[000, 011], [000, 111]}, which is the tight bound of them.

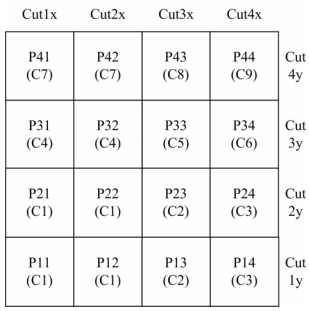

Figure 6. Pointer array in HyperCuts. HyperCuts builds for each of the 4*4 sub-spaces a pointer. The pointers are indexed by the original cuttings rather than the aggregated cuttings.

Each time a packet arrives, the decision tree is traversed based on information in the packet header to reach a leaf node, where a small number of matching rules are stored for linear search. Although this basic structure is similar to the work in [1] [7], ExCuts employs novel ideas that significantly optimize the shape of the decision tree.

4.1 Space Partition with Fixed Number of Cuttings

In HiCuts and HyperCuts, the process to partition the search space at each internal node includes two steps: (1) Identifying the most suitable set of dimensions to partition and (2) determining the number of partitions to be done in each of the chosen dimensions. Both of the two algorithms use a number of sophisticated heuristics and optimizations to choose a proper number of cuttings for each internal node and hence build variable-stride tries as the search structure.

Different from HiCuts and HyperCuts, ExCuts leaves out the process to choose the number of cuttings and set for each internal node a fixed number of cuttings, which results in a fixed-stride trie as the search structure. Experiment results show that when stride w varies from 4 to 7, the total memory space occupied by the search structure varies within the same order of magnitude. To guarantee an ideal worst-case search time, we set stride w=6.

Having a fixed number of cuttings, the only challenge left for space partition process is to pick the most discriminative dimensions which will lead to as few as possible number of child nodes and as less as possible number of rules in each child node. To the best of our knowledge there is no consummate method of picking the most suitable dimensions. A number of local optimized solutions in existing work can be used in ExCuts:

- Minimizing $\max_j(NumberRules(child_j))$ in an attempt to decrease the worst-case depth of the tree. This simple heuristic is suggested in [1]. While easy to implement, it has proved to be ineffective in our experiment.
- Minimizing $\text{avg}_j(NumberRules(child_j))$ in an attempt to decrease the overall memory usage. We suggest using this heuristic in ExCuts because it has proved to be more effective than other heuristics and also very easy to implement.
- Maximizing the number of unique sub-sets of rules in an attempt to search for the most uniform distribution of the rules in child nodes. This heuristic is used in HyperCuts. Although it is effective compared to other methods, searching for unique sub-sets is time consuming in multiple dimensions.

Once the number of cuttings is set and the dimensions are chosen, the current search space is uniformly partitioned in to equal-sized 2^{wf} sub-spaces, where w is the fixed stride and d is the number of dimensions on which the partitions are to be executed. From the view of spatial projection, F-dimensional hyper-rectangles (rules) are projected to the f-dimensional projection-space according to the first w bits along each chosen dimensions of the current search space.

4.2 Two-step Search Space Aggregation

Because the number of child nodes has a direct relationship to the memory space occupied by the search structure, it should be reduced as much as possible. Both HiCuts and HyperCuts use a set of heuristics to try to maximize the reuse of child nodes. Because they only merge adjacent child nodes which have associated with them the same set of rules, only contiguous regions covered by the child nodes are aggregated. However, discontiguous regions may also share the same set of rules. To minimize the number of child nodes, we should aggregate all the sub-spaces sharing the same set of rules. ExCuts uses the following two-step space aggregation process to optimize the reuse of the child nodes.

a) STEP I: Contiguous Space Aggregation

After space partition, many child nodes share identical rules. The first step is to aggregate contiguous sub-spaces along each of the d dimensions. ExCuts employs a series of elegantly designed data-structures to reduce the memory usage.

In HiCuts and HyperCuts, the direct way to traverse the decision tree is to create a pointer array with 2^{wf} pointers, and each of them points to a corresponding child node. If the search space is partitioned along f dimensions at each internal node, the pointer array requires $\Theta(2^{wf})$ memory storage. For example, if there are 2^6 cuttings in 2 dimensions, the internal node needs to store about 4k 32-bit pointers. To avoid memory blow-up, both HiCuts and HyperCuts use variable-strides to limit the number of cuttings at each internal node. However, the stride is a constant in ExCuts, so building such pointer arrays will consume a lot of memory storages. Fortunately, in most of the cases, these pointer arrays can be compressed using the following data-structure:

- Assign *Aggregated Cuttings* (AC) with incremental *space IDs*, from 0 to m_i, i=1, ...,f.
- Create a $\prod_{i=1}^{f} m_i$ -entry *Compressed Pointer Array* (CPA).
- Create for each of the f dimensions a 2^w-entry *Aggregating Bit String* (ABS). Set the bit if the corresponding cutting is not aggregated with the previous one.

When a packet P falls into sub-space (Cut_1, Cut_2, ..., Cut_f), the offsets of the corresponding pointer along each of the f dimensions in CPA can be obtained by adding the first Cut_i bits in ABS ($1 \le i \le f$). Thus ExCuts compresses the 2^{wf}-entry PA into the $\prod_{i=1}^{f} m_i$ -entry CPA. Since m_i ($1 \le i \le f$) is always much smaller than 2^w (due to the contiguous space aggregation), the memory space occupied by the search structure is significantly reduced.

b) STEP II: Discontiguous Space Aggregation

To further eliminate spatial redundancy, ExCuts aggregates discontiguous sub-spaces by setting pointers in CPA to child nodes with distinct set of rules. Because the search space to partition in any internal node must be a contiguous hyper-rectangle, aggregation of discontiguous sub-spaces seems infeasible. Fortunately, we can find a *big enough* hyper-rectangle that tightly contains all the sub-spaces to aggregate, and associate the child node this with hyper-rectangle as the corresponding search space.

Actually, experimental results show that, in most of the cases, non-rectangle regions do not need further partitions.

4. 3 Packet Search Structure

Like HiCuts and HyperCuts, ExCuts works by carefully preprocessing the rulesets to build a decision tree data structure. Each time a packet arrives, the decision tree is traversed down to a certain leaf node, where a small number of rules are stored. Linear search among these rules yields the final matching result. We use a flowchart to describe how to build the packet search structure for of ExCuts. Figure 7 depicts how to build the ExCuts tree.

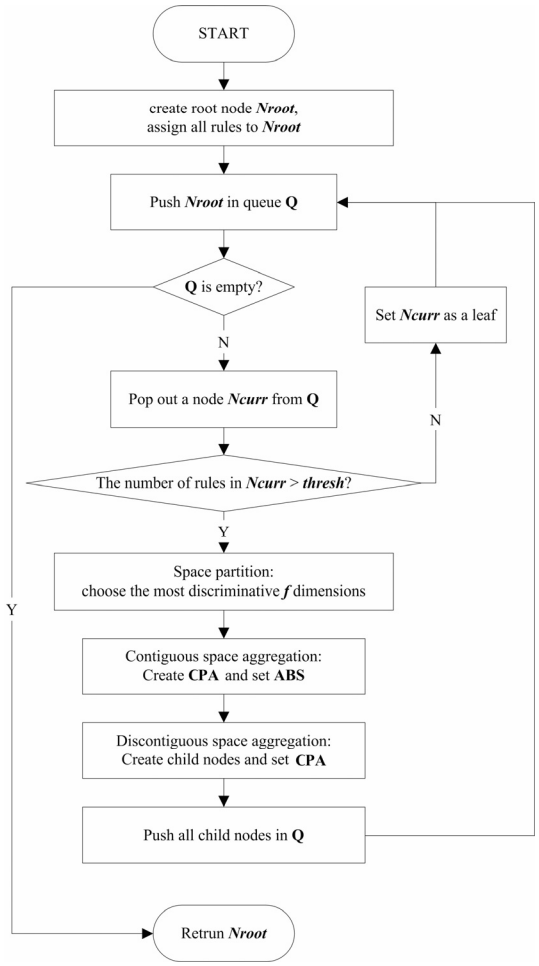

Figure 7. Flowchart of Building the ExCuts Tree.

5. Experimental Results

5.1 Rulesets and Metrics

We evaluate ExCuts both on real-life firewall and core router rulesets as well as on synthetic rulesets. The real-life rulesets are obtained from large enterprise networks and major ISPs. Firewall rulesets are named FW1, FW2, FW3; Core router rulesets are named CR1, CR2, CR3 and CR4. Synthetic rulesets include synthetic firewall sets (SF1~SF20) and synthetic core router sets (SC1~SC10). The largest real-life ruleset (CR4) contains 1945 rules, and the largest synthetic ruleset contains 10000 rules. All rules are 5-dimensional with 32-bit source/destination IP addresses represented as prefixes, 16-bit source/destination port numbers represented as ranges and 8-bit transport layer protocol. It is reported in [3] [7] [8] and [18] that the structural characteristics of firewall policy tables (rulesets) are different from core router access control list (ACL, also rulesets), e.g. most source port ranges in core routers are [0, 65535], while in firewall policy tables source port ranges are assigned more specifically.

Table 1. ExCuts with Different Strides from 4 to 7

Rulesets	No. Rules	ExCuts-4	ExCuts-5	ExCuts-6	ExCuts-7
FW1	68	1,805	1,928	1,316	1,199
FW2	136	3,630	3,876	2,643	2,423
FW3	340	9,150	9,924	6,732	6,287
CR1	500	8,239	7,325	6,189	5,641
CR2	1,000	54,211	45,973	32,693	36,533
CR3	1,530	44,097	42,297	38,984	35,564
CR4	1,945	69,195	65,480	58,105	51,007

(Unit: 32-bit memory word)

Table 2. Memory usage comparison: ExCuts vs. HiCuts & HyperCuts

Rulesets	No. Rules	HiCuts	HyperCuts	ExCuts
FW1	68	5,443	35,401	1,316
FW2	136	10,779	69,782	2,643
FW3	340	24,645	172,932	6,732
CR1	500	29,409	89,005	6,189
CR2	1,000	979,736	871,541	32,693
CR3	1,530	13,606,858	480,225	38,984
CR4	1,945	5,928,724	672,442	58,105

(Unit: 32-bit memory word)

Table 3. Worst-case search time comparison: ExCuts vs. HiCuts & HyperCuts

Rulesets	No. Rules	HiCuts	HyperCuts	ExCuts
FW1	68	30	24	20
FW2	136	32	24	20
FW3	340	32	24	20
CR1	500	40	26	24
CR2	1,000	52	26	24
CR3	1,530	64	28	24
CR4	1,945	60	28	24

(Unit: Memory Access)

Although the number of sets available to us is quite small, we believe that experimental results obtained using real-life rulesets are more convincing than those tested on synthetic rules. Actually, we use the synthetic rulesets just to test the stability of algorithms. Different synthetic rulesets in our experiments have same statistic characteristics, but differ in size.

To test the performance of all the algorithms on both real-life and synthetic rulesets, we examine, for each ruleset, the number of memory accesses as the *search time* (Time) and the amount of *memory usage* (Space) for the whole data structure for search built by the algorithms. Different from [7] [8] (where one memory access is a single 32-bit word access), one memory access here refers to reading a certain number (1~8) of continuous memory words. This metric is in accordance with the experimental evaluation in paper [19].

5.2 Variation with Stride *w*

Table I shows the memory usage of the search structure build by ExCuts with different choice of stride *w*. It can be seen from the table that the memory usage is not sensitive to the choice of *w*. In most of the cases, a larger *w* even leads to smaller memory usage. This seems to be a contradiction to the conclusion made in HiCuts and HyperCuts that the number of cuttings is in direct proportion to the memory usage. In fact, such an increase of memory usage results from increase of the pointer arrays. However in ExCuts, the pointer arrays are efficiently compressed. Thus the total memory usage is just in proportion to the number of tree nodes. Because the choice of a larger *w* implies more particular space decomposition, larger number of cuttings can eliminate some intermediate tree nodes and generate an overall decision tree with fewer nodes. So the memory usage decreases with a larger *w*. To guarantee an ideal worst-case search time, we set stride *w*=6. Too large strides will cause slow preprocessing speed due to the aggregation in multidimensional cuttings.

5.3 Performance on Real-life Rulesets

We first compare ExCuts with the algorithms of the best reported performance, including HiCuts, HyperCuts, RFC and HSM, on real-life rulesets. Due to patent issues, we were not able to obtain source codes from the authors and thus the codes of these algorithms are all written by ourselves. We make our best effort to make sure the fairness of our result analysis. Experimental results show that our codes achieved nearly the same performance compared to the experimental results reported in [7].

In comparison with algorithms using decision trees, Table II and Table III compare ExCuts with HiCuts & HyperCuts respectively on spatial and temporal performance. We can see from these tables that, for all real-life rulesets, ExCuts achieves at least an order of improvement in memory usage, as well as superior worst-case search time.

To compare with the algorithms using lookup tables, Table IV shows the memory usage of RFC, HSM and ExCuts. For the largest real-life ruleset CR4, ExCuts uses 16 times less memory than HSM and 27 times less than

Table 4. Memory usage comparison: ExCuts vs. RFC & HSM

Rulesets	No. Rules	RFC	HSM	ExCuts
FW1	68	200,652	10,223	1,316
FW2	136	209,602	27,657	2,643
FW3	340	296,382	65,581	6,732
CR1	500	264,987	29,814	6,189
CR2	1,000	530,539	230,716	32,693
CR3	1,530	863,476	486,857	38,984
CR4	1,945	1,580,005	989,161	58,105

(Unit: 32-bit word)

Table 5. Worst-case search time comparison: ExCuts vs. RFC & HSM

Rulesets	No. Rules	RFC	HSM	ExCuts
FW1	68	9	17	20
FW2	136	9	17	20
FW3	340	9	21	20
CR1	500	9	23	24
CR2	1,000	9	25	24
CR3	1,530	9	25	24
CR4	1,945	9	25	24

(Unit: Memory Access)

RFC. Although the search speed of RFC and HSM are 20%~120% faster that of ExCuts (see Table V), they may require parallel searches while ExCuts can be fully pipelined to implement for fast search.

5.4 Performance on Synthetic Rulesets

In order to compare the ability to handle large number of rules, we test ExCuts and HyperCuts on a series of synthetic rulesets. Although we believe that tests on real-life rulesets are more persuasive, the task is made harder because the real-life rulesets available to us is quite limited. To guarantee the fairness of experimental comparison, we create these rulesets in the same way as the synthetic rules in [7], and all of these rules have identical distribution at each field.

Table VI shows the test on synthetic firewall rulesets SF1~SF20 (SF1 has 100 rules and SF20 has 2000 rules). We see that both HyperCuts and ExCuts perform stably with the number of rules less than 1700. But when the number of rules becomes larger (than 1700), the memory usage of HyperCuts has a sharp increase in comparison with that of ExCuts. The total memory used by HyperCuts on SF20 is 10 times larger than that on SF17, while ExCuts on SF20 uses no more than 30% more memory as on SF17. Similar results are obtained when we test other algorithms on these synthetic rulesets. ExCuts is proved to be more stable than all other algorithms we have tested. Such a conclusion is also supported by the results in Table I and Table II with real-life rulesets.

Table VII shows the experimental result on large synthetic core router sets SC1~SC10 (SC1 has 1000 rules and SC10 has 10000 rules). The results prove that the memory space occupied by ExCuts scales linearly in number of rules. However, just like F. Baboescu said in [8], this should be taken with a grain of salt because the large ruleset generation methodology preserves the source-destination structure of the original real-life ruleset. If this assumption does not hold as rulesets scale up, ExCuts may not linearly scale. However, it is not sure that if there will be such complicated real-life routers in near future.

6. Conclusion

Packet classification has received tremendous attention in recent years. While hardware like Ternary CAMs offers a good solution for small rulesets, they may use too much power and board area for large rulesets. Thus, it is worth looking for alternatives to overcome the limits in hardware solutions, and the challenge of finding efficient algorithmic approaches for packet classification to achieve high performance with comparatively low hardware requirement still motivates the research today.

Although theoretical bounds tell us that it is not possible to arrive at a practical worst-case solution, real-life rulesets have characteristics that can be exploited in algorithms to generate different search structures for various kinds of packet classification applications. Thorough studies in theoretical analysis, as well as new observation in data characteristics, are examples of research effort in designing more efficient and practical packet classification algorithms. Despite the vast amount of genius ideas explored in prior work, there are still novel ideas springing out together with the advancing of theories and technologies, which can further improve the performance of existing best algorithms.

Recently, pioneering work on packet classification based on decision trees and geometric cuts provide high search rate using modest memories. As an extension of the best-known existing decision tree scheme HyperCuts, the proposed algorithm ExCuts significantly improves the performance of existing decision tree algorithms in terms of both search rate and memory usage. First, ExCuts refines the space aggregation step in HyperCuts with a discontiguous space aggregation scheme, which greatly reduces the number of tree nodes; Secondly, ExCuts adopts a bit string to compress the size of the large pointer

Table 6. Memory Usage Comparison on Synthetic Firewall Rulesets: ExCuts vs. HyperCuts

Rulesets	No. Rules	HyperCuts	ExCuts	Rulesets	No. Rules	HyperCuts	ExCuts
SF1	100	1,661	837	SF11	1,100	19,551	8,896
SF2	200	2,109	1,485	SF12	1,200	20,162	10,234
SF3	300	2,396	1,972	SF13	1,300	22,335	11,225
SF4	400	3,630	2,400	SF14	1,400	22,456	12,084
SF5	500	3,826	2,800	SF15	1,500	23,009	13,939
SF6	600	5,060	3,230	SF16	1,600	25,586	13,637
SF7	700	11,526	3,870	SF17	1,700	29,044	14,522
SF8	800	12,068	4,581	SF18	1,800	57,612	15,310
SF9	900	17,096	5,749	SF19	1,900	78,479	16,030
SF10	1,000	18,301	7,007	SF20	2,000	286,010	17,898

(Unit: 32-bit memory word)

Table 7. Memory Usage on Synthetic Core Router Rulesets

Rulesets	No. Rules	ExCuts	Rulesets	No. Rules	ExCuts
SC1	1,000	19,632	SC6	6,000	117,914
SC2	2,000	39,317	SC7	7,000	137,573
SC3	3,000	58,991	SC8	8,000	157,294
SC4	4,000	78,646	SC9	9,000	176,812
SC5	5,000	98,298	SC10	10,000	196,578

(Unit: 32-bit memory word)

arrays in internal nodes. Finally, both of these techniques make it feasible and practical for ExCuts to pick a fixed number of cuttings at each internal node, and hence provide ExCuts with an explicit worst-case search time.

Experimental results show that ExCuts outperforms the best result of existing heuristic algorithms on both real-life rulesets and synthetic classifiers. Compared to the best-known decision tree algorithms, ExCuts uses 11 to 27 times less memory storage and 10% to 50% less time in worst-case search. ExCuts also outperforms other popular algorithms such as RFC and HSM with best time/space tradeoffs.

Acknowledgements

This work is sponsored by the Intel IXA University Program.

References

[1] P. Gupta and N. McKeown, Packet classification using hierarchical intelligent cuttings, *Proc. Hot Interconnects,* 1999.

[2] M.H. Overmars and A.F. van der Stappen, Range searching and point location among fat objects, *Journal of Algorithms, 21(3)*, 1996, 629-656.

[3] P. Gupta and N. McKeown, Packet classification on multiple fields, *Proc. ACM SIGCOMM*, 1999, 147~160.

[4] B. Xu, D. Jiang, and J. Li, HSM: A fast packet classification algorithm, *Proc. 19th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, Taiwan, 2005, 1: 987-992.

[5] V. Srinivasan, G. Varghese, S. Suri and M. Waldvogel, Fast and scalable layer four switching, *Proc. ACM SIGCOMM*, 1998, 191-202.

[6] F. Baboescu and G. Varghese, Scalable packet classification, *Proc. ACM SIGCOMM*, 2001, 199-210.

[7] S. Singh, F. Baboescu, G. Varghese and J. Wang, Packet classification using multidimensional cutting. *Proc. ACM SIGCOMM*, 2003, 213-224.

[8] F. Baboescu, S. Singh and G. Varghese, Packet classification for core routers: Is there an alternative to CAMs? *Proc. IEEE INFOCOM*, 2003, 1:53-63.

[9] V.Srinivasan, S.Suri and G.Varghese, Packet classification using tuple space search, *Proc. ACM SIGCOMM*, 1999, 135-146.

[10] J. van Lunteren and T. Engbersen, Fast and scalable packet classification, *IEEE Journal on Selected Areas in Communications 21(4)*, 2003, 560-571.

[11] A. Feldman and S. Muthukrishnan. Tradeoffs for packet classification, *Proc. IEEE INFOCOM*, 2000, 3: 1193-1202.

[12] T. Lakshman and D. Stiliadis, High speed policy-based packet forwarding using efficient multi-dimensional range matching, *Proc. ACM SIGCOMM*, 1998, 203-214.

[13] T.Y.C Woo, A modular approach to packet classification: algorithms and results, *Proc. IEEE INFOCOM*, 2000, 3:1213-1222.

[14] F. Geraci, M. Pellegrini and P. Pisati, Packet classification via improved space decomposition Techniques, *Proc. IEEE INFOCOM*, 2005, 1:304-312.

[15] Y. Qi and J. Li, Dynamic cuttings: packet classification with network traffic statistics, *3rd Proc. International Trusted Internet Workshop*, 2004.

[16] P. Gupta and N. McKewon, Algorithms for packet classification, *IEEE Network 15(2)*, 2001, 24-32.

[17] D. E. Taylor, Survey and taxonomy of packet classification techniques, *ACM Computing Surveys 37(3)*, 2005, 238-275.

[18] M.E. Kounavis, A. Kumar, H. Vin, R. Yavatkar and A.T. Campbell, Directions in packet classification for network processors, *Proc. 2nd Workshop on Network Processors*, 2003.

[19] Y. Qi, B. Xu and J. Li, Performance evaluation and improvement of algorithmic approaches for packet classification, *Proc. International Conference on Network and Services*, 2005.