Limitations of the Hierarchical Intelligent Cuttings Packet Classification Algorithm

Tyrell Sassen, Neco Ventura
Department of Electrical Engineering, University of Cape Town,
Rondebosch, Cape Town, South Africa

{tsassen, neco}@crg.ee.uct.ac.za

Abstract—The packet classification problem is a complex problem to solve, requiring either a high storage capacity or high query time. Most current algorithms implement a heuristics-based approach, which exploits the structure of real-world rulesets to simplify the problem complexity. The HiCuts algorithm uses a number of metrics in order to create its decision tree structure. Several of the metrics suggested in the literature are impractical as they fail to classify rulesets with particular structures. This paper identifies these failings and suggests a number of improvements.

Index Terms—Packet Classification, Differentiated Services, Quality of Service

I. Introduction

IN the last few years, we have seen the internet move away from its traditional role of email and file transfer, and start to become a more unified network, dealing with a variety of realtime and non-realtime data. While IP was specifically designed as a 'best-effort' protocol, not providing any QoS guarantees, it is now needing to be adapted to provide data delivery bounds for realtime traffic like voice and video.

There has been a great deal of research into this area and a number of different protocols have been developed to provide QoS over IP, one of the more common protocols is known as the diffserv protocol [3]. The aim of diffserv is extend the TCP protocol to provide a field in which the QoS requirements of the traffic stream are encoded. This will allow traffic with higher service requirements to be given preferential treatment at each router hop along its path. In order to provide these different levels of service, it is necessary to examine the incoming packet fields and determine to which traffic class the packets belong. This process of examining and marking the packets is known as packet classification.

The need to provide packet classification and other processing intensive functions, such as packet scheduling and buffer management, has identified another area of network functionality that needs to be addressed, namely intelligent, data-processing network routers.

The authors would like to thank Telkom SA, Siemens, Intel, the National Research Foundation (NRF) and the Department of Trade and Industry (DTI) for supporting this research project.

This need for more powerful, flexible network entities has lead to the development of soft-programmable network processors (NPs) specifically designed to process packets at line-speeds. These NPs provide the flexibility of general-purpose processors with the speed of more traditional Application-Specific Integrated Circuits (ASICs).

NPs are ideal for implementing diffserv-enabled routers as they provide packet-processing engines that have sufficient processing power to classify packets at high line speeds, while still allowing for a great deal of flexibility in terms of the algorithms used.

II. Introduction to Packet Classification

Although the issue of packet classification has been dealt with many times in the past, it is still an area of study that holds a great deal of room for improvement. The packet classification problem is inherently a hard problem to solve with worst-case bounds being found in the area of computational geometry; the problem of finding the enclosing region of a point in an F-dimensional space, given n multidimensional regions. This problem can be either optimised to minimise query time, in which case the storage requirements for the algorithm become very large, or optimised to minimise storage space, which in turn leads to very large query times [1]. The exact orders of these optimisations are listed below:

1) If the problem is optimised for query time it is possible to resolve a query in $O(\log n)$, with a storage complexity of $O(n^F)$.
2) In order to optimise for storage complexity, the storage complexity drops to $O(n)$, with an average query time of $O(\log^{F-1} n)$.

While these orders of magnitude could be acceptable for a small number of rules, current routers can contain many thousands of classification rules [2]; and this number is only expected to increase as protocols like diffserv [3] provide an increased level of service differentiation.

In the last few years, there have been a number of papers published [4][5][6][7][8] that try to provide a practical means of solving this problem. Many of these papers suggest a heuristics-based approach, showing that current classification rule-sets have an inherent structure that is possible to exploit, allowing the problem to be solved much easier than suggested by the theoretical bounds mentioned in [1]. NPs are ideal for these kinds of conditions as they

provide the flexibility to choose the classification algorithm based on the specific structure of the ruleset being classified.

Two algorithms that use heuristics have been suggested by Gupta *et al.* The packet classification algorithm called *Recursive Flow Classification* (RFC) [6], was suggested first, but this algorithm still suffered from relatively high storage requirements. In response to this the authors suggested a new algorithm known as *Hierarchical Intelligent Cuttings* (HiCuts) [7]. The HiCuts algorithm offers reduced storage requirements, when compared to the RFC algorithm, though this depends largely on the structure of the rule set used. These limitations will be explored more in Section III.

TABLE I
EXAMPLE RULESET

Rule	*x* Range	*y* Range
R1	0 – 31	0 – 255
R2	0 – 255	128 – 131
R3	64 – 71	128 – 255
R4	67 – 67	0 – 127
R5	64 – 71	0 – 15
R6	128 – 191	4 – 131
R7	192 – 192	0 – 255

Both the *x* and *y* dimensions cover an 8-bit range, with a minimum value of 0 and a maximum of 255. All ranges are inclusive of endpoints.

III. OVERVIEW OF THE HICUTS ALGORITHM

The HiCuts algorithm [7] seeks to break up the packet classification task into two tasks: the first task being a decision tree traversal to find the particular *n*-dimensional area in which the packet fits. The second task is performing a linear search on a small number of rules, at the leaf nodes of the decision tree, to provide an exact rule match.

The HiCuts algorithm is highly dependent on a pre-processing stage that is responsible for building the decision tree. This stage is responsible for the depth and shape of the final tree and is tuned by a number of parameters, namely the rule threshold (*binth*) and the space measure factor (*spfac*).

The parameter *binth* determines the maximum number of rules to hold at each leaf node. If the number of rules at the node is greater then *binth*, then the depth of the tree is increased and the node is split again using the same recursive algorithm. The second parameter, *spfac*, is responsible for determining the number of cuts to make at a particular node. Each time a node is set to be cut, a space measure function (smf) is used to approximate the amount of storage space required to store the node. The number of cuts is doubled as long as the space measure function is below a threshold function that is defined as proportional to *spfac*. This ensures that the storage requirements for the node do not outweigh the benefits of cutting the node in the first place.

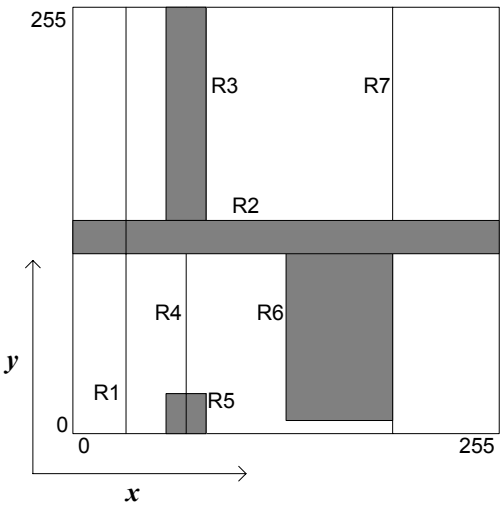

Fig. 1. The example ruleset from Table I expressed graphically.

Although HiCuts is clear about the heuristic used to determine the number of cuts at each node, it is less clear about the heuristic used to determine which dimension to cut along in each node. Since the HiCuts algorithm only allows a cut to be made in one dimension at a time, the choice of dimension will have a large impact on the depth of the tree. The authors do not provide an exact mechanism for choosing the dimension in which to cut, though they suggest a number of different metrics that could be used.

These are listed below:

1) For the *j* child nodes, minimise $\max_j(\text{numrules}(\text{child}_j))$ in order to decrease the worst-case depth of the tree.
2) Treat $\text{numrules}(\text{child}_j)/\text{totalnumrules}$ as a probability distribution and maximise the entropy of the distribution. This attempts to pick the dimension that will lead to the most uniform distribution of rules across nodes.
3) Choose to cut the dimension that will require the least amount of storage space, i.e. minimise *smf* over all dimensions.
4) Cut the dimension with the largest number of distinct rule components, e.g. if two rules share the same exact source IP address range, this is considered one distinct component.

In order to show more clearly, the workings of the HiCuts algorithm, the example ruleset in Table I, shown graphically in Fig. 1, is given. For *spfac* = 2 and *binth* = 2, this ruleset maps to the tree shown in Fig. 2. This tree contains five leaf nodes and two parent nodes. Each leaf node contains two rules, satisfying the *binth* parameter of the tree. Each parent node is defined by three variable, these are shown in the parentheses next to each node. The first value shown is the size of the region that this node covers. As is expected, the root node covers the whole region (256x256), while the second parent node only covers a 4th of the parent region. This is intuitive as the root node has four children, which the region is split amongst equally. The second parameter defines in which dimension the cut occurs. The cuts are parallel to the *x*-dimension in the root node and parallel to the *y*-dimension in the second parent node. The last parameter is each set of parentheses defines the number of cuts that are made at each parent node.

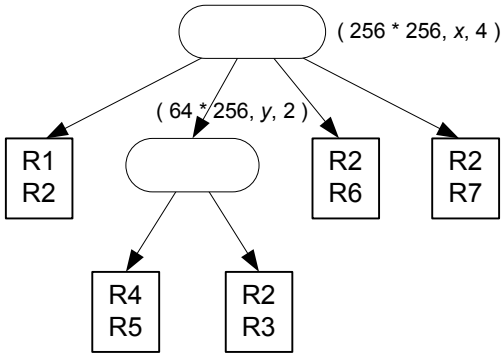

Fig. 2. HiCuts tree created from the ruleset define in Table I using the parameters *spfac* = 2 and *binth* = 2.

IV. Limitations of HiCuts Heuristics

A. Dimension Cutting Metrics

Gupta *et al* suggest the four metrics listed in Section II as possible heuristics that could be used to determine in which dimension to make the cuts. This section will show that, in some cases, the metrics numbered 1, 2 and 3 are inadequate; this will be done by means of a simple 2-dimensional example, which will later be extended to cover the *n*-dimensional case.

The problem ruleset that will be used to show these inadequacies is defined in Table II. This ruleset is also shown graphically, as a series of shaded regions, in Fig. 3. The dashed lines represent the number of cuts made in each dimension, for *spfac* = 2 and *binth* = 3.

TABLE II
Problem Ruleset

Rule	*x* Range	*y* Range
R1	228 – 231	0 – 255
R2	236 – 239	0 – 255
R3	244 – 247	0 – 255
R4	252 – 255	0 – 255

Both the *x* and *y* dimensions cover an 8-bit range, with a minimum value of 0 and a maximum of 255. All ranges are inclusive of endpoints.

Let us consider metric number 1. It can be seen from Fig. 3 that the maximum number of rules for any child, for the cut parallel to the *x*-axis, is 4, since this is the number of rules in both child nodes. This is also true for the cuts parallel to the *y*-axis since the last child contains all 4 rules, while the others contain none. Since both these values are the same, it would be conceivable that the algorithm would choose the first dimension with the smallest value, the *x*-dimension in this case. This causes a problem in that, even though the node has been cut, the number of rules in each node has not been decreased. We are therefore left with two child nodes, each similar in structure to the parent node. It is not hard to see that if the same metric is used to determine the dimension for cutting at these child nodes, the nodes will be split once again in the same dimension. The nodes will never have a reduced number of rules and therefore never become leaf nodes, causing the nodes to be split an infinite amount of times or until the algorithm runs out of memory. This is shown graphically in Fig. 4.

Although the second metric approaches the problem in a completely different way, it leads to the same result. Metric 2 tries to create a balanced decision tree by picking the dimension with the most uniform rule distribution[1]. This causes a problem in that the dimension with the most uniform distribution is not necessarily the dimension that will cause the number of rules in each child to decrease. In fact for the example case in Fig. 3, this is exactly what happens. For the cut parallel to the *x*-axis the number of rules in each child is equal, leading to a uniform rule distribution, while for the cuts parallel to the *y*-axis all the rules are clustered in one child, leading to a very uneven distribution of rules. The algorithm will therefore pick the first-dimension and will in turn lead the same problem as seen in Fig 4.

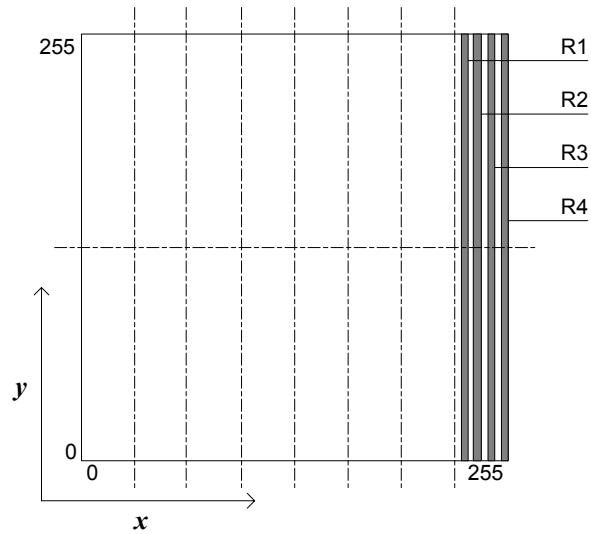

Fig. 3. The problem ruleset from Table II expressed graphically. The shaded regions represent the rules and the dashed lines show the cuts to be made in either dimension.

In order to show how the third metric fails, the exact mechanism for determining the space measure must first be explained. The space measure function is calculated by adding the number of cuts made in the particular dimension to the total number of rules in each child. For our example this leads to *smf* = 9 for the first dimension and *smf* = 11 for the second[2]. Choosing the smallest *smf* will lead to the cut being made in the first dimension once again. This phenomenon becomes more common as the number of cuts becomes larger. For dimensions with large ranges and a small number of clustered rules, the number-of-cuts term of the *smf* will tend to outweigh the number-of-child-rules term leading to the dimension with the smaller number of cuts being chosen, regardless of the rule distribution.

The fourth metric described in the literature does not share the problem of the first three. Choosing the dimension

[1] The equation used to calculate the entropy of a discrete distribution can be found in [9]

[2] The second dimension has more cuts than the first, because the number of cuts is doubled each time the *spf* is below the threshold, which is calculated by multiplying the *spfac* with the total number of rules.

with the largest number of distinct rule components will intuitively choose the dimension with the most potential for decreasing the number of rules in its child nodes. This avoids the problem with the above metrics by allowing the nodes to be split enough, so that they eventually become leaf nodes. This can be seen in our example case, where the number of distinct components in the *x*-dimension is 4 and the number in the *y*-dimension is 1, as all rules share the same range: 0 – 255. Our experiments showed that this is the only one, of the four metrics, that could classify all commonly occurring rulesets [10].

Using the 'distinct component' metric, we were able to successfully produce HiCuts trees containing over 10 000 rules. All other metrics failed to produce decision trees when the number of rules increased above 200. This was caused by the different metrics not being able to subdivide a particular node, and therefore causing the algorithm to enter a continuous loop. The large decision trees, produced using the 'distinct component' metric, revealed a second problem with the HiCuts algorithm that is discussed in section B.

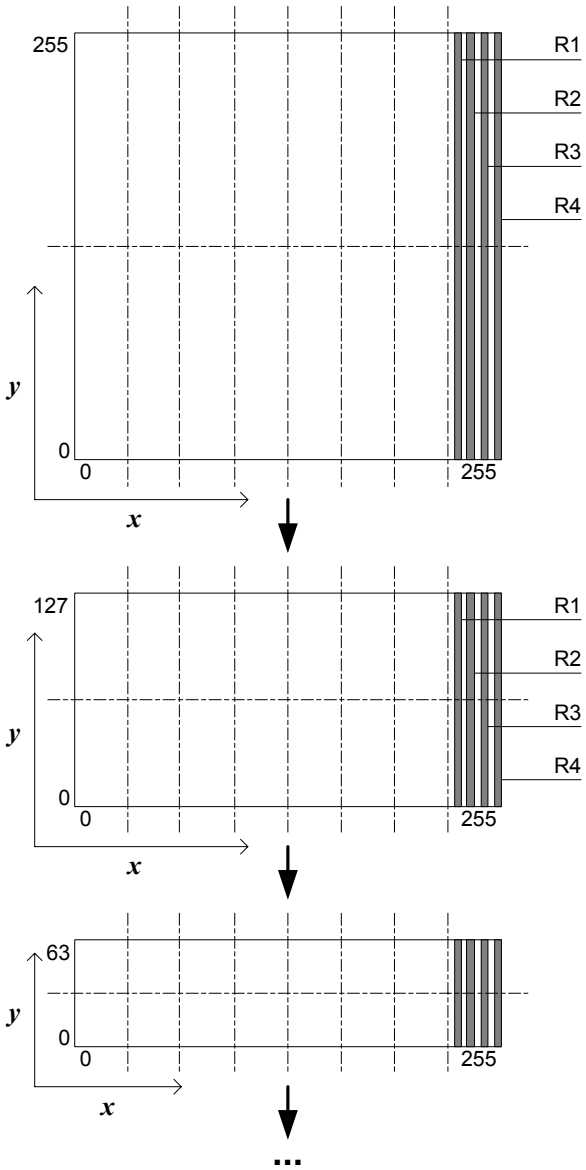

Fig. 4. As the region is cut parallel to the *x*-axis, the size of the region is reduced but the number of rules in the region remains the same.

In this section we have shown how the choice of heuristic can have an impact on the effectiveness of the classifier. Although this has only been shown for a trivial classifier, the ideas introduced can be extended to *n*-dimensional classifiers as they do not depend on the number of dimensions but rather on the layout of the rules themselves. The problem is most apparent when working with dimensions that have large ranges with at least *binth* + 1 rules clustered in a very small area. Research has shown that the problems identified in this example can be extended to even a 5-dimensional classifier working with rulesets commonly found in Internet firewalls [10]. One possible solution to this problem is to always use the fourth metric when implementing the HiCuts algorithm.

B. Rule Threshold Parameter

The second major problem with the HiCuts algorithm is a product of the *binth* tuning parameter. The algorithm states that if a node contains more than *binth* rules, the node must be split. If there are at least *binth* + 1 rules at a node, and these rules span the entire *n*-dimensional region of the node, it is impossible to split the node in order to decrease the number of rules at the children. No matter in which dimension the cut occurs, all children of the node will have the same number of rules as the parent. This leads to the same problem as seen above, where the algorithm will continue to cut nodes until it eventually runs out of memory.

This problem stems from the fact that the parameter *binth* is set by the user without consideration of the classifying ruleset, it might be possible to avoid this problem if *binth* is scaled automatically depending on the structure and amount of overlap in the ruleset. This could lead to further problems where *binth* is increased enough that the linear search portion of the algorithm dominates the query time, hindering the efficiency of the classifier.

V. Conclusion

This paper has described several problems that exist with the HiCuts packet classification algorithm. These problems stem from either a bad choice of 'dimension choosing' metric, or a bad choice of the rule threshold parameter *binth*. The authors note that these problems only arise when the classifying ruleset has a particular structure, though this structure is common enough in real-world firewalls to warrant concern. The problems described in this paper can be avoided, for the most part, by using the 'distinct component' metric, outlined in the literature, and choosing a large enough value for *binth*.

References

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

[2] F. Baboescu, S. Singh and G. Varghese, "Packet classification for core routers: Is there an alternative to CAMs?", *INFOCOM*, 2003.

[3] Differentiated Services (diffserv) Charter, http://www.ietf.org/html.charters/diffserv-charter.html, September 2002.
[4] T. Lakshman and D. Stiliadis, "High speed policy-based packet forwarding using efficient multi-dimensional range matching", *SIGCOMM*, 1998.
[5] V. Srinivasan, S. Suri and G. Varghese, "Packet Classification using Tuple Space Search", *SIGCOMM '99*, 1999.
[6] P. Gupta and N. McKeown, "Packet classification on multiple fields", *SIGCOMM 99*, 1999.
[7] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings", *Proc. Hot Interconnects*, 1999.
[8] S. Singh *et al*, "Packet classification using multidimensional cutting", *SIGCOMM '03*, 2003.
[9] Principle of Maximum Entropy, Wikipedia, http://en.wikipedia.org/wiki/Principle_of_maximum_entropy, April 2005.
[10] D.E. Taylor and J.S. Turner, "ClassBench: A packet classification benchmark", *INFOCOM*, 2005.

BIOGRAPHIES

Tyrell Sassen (tsassen@crg.ee.uct.ac.za) is an MSc student in Electrical Engineering at the University of Cape Town, South Africa. He got his B.Sc (Eng) degree in Electrical and Computer Engineering from the University, of Cape Town in 2003. His research interests are in Network Processors and Packet Classification.