Packet Classification in Large ISPs: Design and Evaluation of Decision Tree Classifiers

Edith Cohen
AT&T Labs–Research
180 Park Avenue
Florham Park, NJ 07932, USA
edith@research.att.com

Carsten Lund
AT&T Labs–Research
180 Park Avenue
Florham Park, NJ 07932, USA
lund@research.att.com

ABSTRACT

Packet classification, although extensively studied, is an evolving problem. Growing and changing needs necessitate the use of larger filters with more complex rules. The increased complexity and size pose implementation challenges on current hardware solutions and drive the development of software classifiers, in particular, decision-tree based classifiers. Important performance measures for these classifiers are time and memory due to required high throughput and use of limited fast memory.

We analyze Tier 1 ISP data that includes filters and corresponding traffic from over a hundred edge routers and thousands of interfaces. We provide a comprehensive view on packet classification in an operational network and glean insights that help us design more effective classification algorithms.

We propose and evaluate decision tree classifiers with *common branches*. These classifiers have linear worst-case memory bounds and require much less memory than standard decision tree classifiers, but nonetheless, we show that on our data have similar average and worst-case time performance. We argue that common-branches exploit structure that is present in real-life data sets.

We observe a strong Zipf-like pattern in the usage of rules in a classifier, where a very small number of rules resolves the bulk of traffic and most rules are essentially never used. Inspired by this observation, we propose *traffic-aware* classifiers that obtain superior average-case and bounded worst-case performance. Good average-case can boost performance of software classifiers that can be used in small to medium sized routers and are also important for traffic analysis and traffic engineering.

Categories and Subject Descriptors

C.2 [**Communication Networks**]: C.2.6 Internetworking;C.2.3 Network Operations

General Terms

Algorithms,Design,Management,Performance,Security

Keywords

packet filtering; routing; decision trees; access control lists

1. INTRODUCTION

Packet classification is extensively deployed at the core and the edge of the Internet and is a well studied problem (e.g.[18, 24, 14, 15, 25, 11, 20, 16, 3, 2, 21]). Yet, it is still growing in importance and evolving. Classifiers grow in size, complexity, and functionality. New applications drive the development of richer filter specifications: Packet classifiers, traditionally used for basic security, are now used for routing, rate control, and mapping packets to different classes of service with Differentiated Services (Diffserv). Classifiers also grow in size; a single ISP router may need to perform filtering for multiple customers VPNs, each using its own set of rules.

Cisco Access Control Lists (ACL) are the industry standard for packet classification [19, 1]. Basic ACLs use only network addresses (source and destination), Extended ACLs operate on layer 4 information (ports, protocol, type of service, etc).

An ACL is an ordered list of rules (which we refer to as a *filter*), each rule has conditions and action. The action of the filter on a packet is that of the highest priority rule that applies to the packet. Packet classification at high speed core routers is traditionally implemented using special hardware called Trenary CAMs (TCAMs). TCAMs evaluate all rules simultaneously and the hardware outputs the lowest index match. TCAMs can handle the fast line speeds required, but are also expensive, consume a lot of energy, and imposes limits on the size and flexibility of rule specifications (for example, TCAMs do not support range tests very efficiently). For this reason, considerable attention has recently been given to software-based classification[18, 14, 15, 25, 11, 16, 3, 2, 21].

Classification algorithms have different tradeoffs between memory size and worst-case search time. Linear search through the rules is a simple and memory efficient classifier, but the worst-case search time grows linearly in the size of the filter. On the other extreme, proposed fast but memory intensive algorithms include Crossproducting[24], bit-level parallelism[18] and RFC [14]. Decision-tree based classifiers were pioneered by Gupta and McKeown [15] and by Woo [25]. They currently seem to outperform other approaches with superior memory versus time tradeoffs and a simple evaluation procedure that can be pipelined [21].

Software classification is also important for traffic engineering and planning, in a simulator testing the effects of certain classifier on traffic before a particular classifier is implemented. ACLs are used to map traffic to different classes of service for quality of service (QoS) and rate control. The configuration of QoS is complex and relies on a good understanding of the traffic patterns in the various router queues and interfaces. It is thus important to determine, prior to deploying a proposed filter, if the available bandwidth for each class is not exceeded.

Our contributions.

Our work is the first to use actual, comprehensive, filter and traffic data from a large ISP (Previous work was based on handful of filters with no corresponding traffic data). From this data we glean insights that help us design better classification algorithms. We evaluate our algorithms using the actual filter and traffic data, but argue that the properties we exploited are likely to hold for larger filters and more complex rules.

Decision trees with common branches

Decision-tree based classification algorithms use geometric cutting to eliminate rules. Each interior (decision) node has an associated test, and each leaf node contains a subset of the (original) rules. Tests associated with internal node are applied to a packet, when a leaf is reached, linear search is applied to the rules associated with the leaf.

Decision-tree algorithms suffer from extensive use of wildcards in the rules. Wildcards mean that many rules have domains that don't distinctly fall in a single child of each decision node and rules get replicated in multiple children, requiring more memory. We propose *common-branches* decision trees where rules that need to be assigned to more than one child get handled separately. Thus, common-branches trees have *worst-case* size that is *linear* in the original list of rules (whereas size can grow exponentially for standard decision trees). A possible weakness of common-branches is increased search times. We argue, however, using simple models, that even though significantly increased search times can occur on contrived synthetic rule sets, common-branches decision trees have small search times when a certain structure, that is typical to real-life rule sets is present. This structure amounts to correlations between wildcards and specified locations of different rules in the filter.

On our data, common-branches trees were only a fraction larger than the original list of rules and used considerably less memory than standard decision trees. Worst-case and average-case search times, however, were comparable. Therefore, common-branches trees clearly outperformed standard trees.

Structure present in rule-sets was exploited by previously proposed classifiers to obtain better tradeoffs between memory and search times. The distinct advantages of common branches trees are very efficient memory, and in particular, the first to offer a linear worst case bound with good search times; are able to exploit rule-set structure implicitly and seamlessly whereas previous algorithms explicitly considered the specified fields in each rule. Moreover, the evaluation procedure itself is very simple, resembling that of standard decision trees: There is no need to maintain extra state and it can be fully pipelined.

Traffic-aware classification algorithms

The tight working constraints of high speed core routers promoted memory and worst-case time as the metrics of interest for IP routing [6, 5, 13, 23] and packet classification [18] algorithms. (IP routing lookup is a subproblem of packet classification where classification is performed only on the destination address field and is strictly prefix based – identifying the rule with the longest matching prefix).

But alongside the traditional metrics of memory size and worst-case time, packet classifiers in some settings can greatly benefit from improved *average-case time* performance. Such settings include classifiers used in software simulations for planning and traffic engineering purposes. With highly stable traffic patterns, classifiers with good average case performance are also applicable to some online packet classification settings. IP route lookup algorithms with good average-case performance were proposed and argued for [6, 17].

Our data reveals a strong Zipf-like pattern where few rules in each filter are responsible to resolving most of the traffic. Moreover, it seems that the bulk of rules is almost never used (also on short time scales). This suggests that traffic-aware algorithms can greatly improve average search times. The evident stability in traffic patterns also support the viability of average-case time algorithms, that are susceptible to sudden shifts in traffic and surges.

We design classification algorithms that are geared towards good average-time performance. We first consider linear search where average time can be improved by reordering the rules in the filter. We then construct decision trees (standard and common-branches) that locally optimize average case time. We evaluate these algorithms on actual filters and packet traffic and demonstrate that average-case performance can be significantly improved over that of classification algorithms that are oblivious to traffic patterns.

We proceed with related work in Section 2, then present properties of our data in Section 3. The algorithms we use are presented in Section 4. Section 5 uses simple models and properties of rule-sets to motivate common branches decision trees. The performance evaluation is presented in Section 6.

2. RELATED PRIOR WORK

Decision tree classifiers are constructed top down in a locally-optimal greedy manner. For each node, local optimization is used to choose the best test to apply or to make the node a leaf.

A heuristic proposed in [15, 25] is to stop the splitting once a node has less than some determined number of rules as a way to reduce memory size, this heuristic is applicable to common branches decision trees as well, and will guarantee that the memory used by the resulting classifier is no more than a fraction larger than the size of the original list of rules. Another effective heuristic to reduce memory is "pushing common rules upwards," where rules common to all descendent leaves are processed at the common ancestor [21] instead of being replicated in all children. This heuristic is applicable to standard decision trees and we implemented it in our constructions to obtain standard decision trees with smaller memory.

Previous work used different tests at decision nodes. Gupta and McKeown [15] introduced Hierarchical Cutting (HiCuts), where the decision at each node is a range test on some field, and Woo [25] used bit tests. Range cuts and bit tests have different advantages. Range cuts work well on the port field (which is specified by ranges) and the protocol field (specified by points). They are also appropriate to address fields with prefix-based masks. Even though the vast majority of address fields in rules are still specified using prefix masks, Extended ACL specs allow for generally placed wildcards. Bit tests can thus provide further flexibility. We used both range and bit tests in our decision trees.

The structure present in real-life rule sets, of which fields are constrained or wildcarded, was exploited by different classification algorithms. Algorithms for basic ACLs took advantage of the fact that often only one of the address fields is specified. One approach is to associate each rule with one of its specified fields; separately match a packet against the rules of each field; and combine the results to find the minimum index matching rule overall [22]. Woo [25] proposed to use a different decision tree for each field. This approach also allows the use of off the shelf existing IP route lookup algorithms for the address fields [18, 11].

The extended grid of tries algorithm (EGT-PC) is motivated by the observation that any given packet matches only a few rules, even when only considering the source-destination prefixes spec-

	action	protocol	src addr and mask	dest addr and mask	port
1	deny	udp	any	18.240.0.0 0.15.255.255	range snmp snmptrap
2	deny	tcp	any	18.240.0.0 0.15.255.255	range 161 162
3	permit	icmp	any	any	ttl-exceeded
4	permit	icmp	any	any	port-unreachable
5	deny	ip	any	18.240.0.0 0.3.255.255	
6	deny	ip	any	18.250.0.0 0.1.255.255	
7	deny	ip	any	18.254.0.0 0.0.7.255	
8	deny	ip	any	18.255.0.0 0.0.15.255	
9	permit	ip	18.1.146.0 0. 0.1.255	any	
10	permit	ip	18.1.16.176 0.0.0.15	any	
11	permit	ip	18.247.13.192 0.0.0.3	any	

Table 1: Example of an 11-rule filter

ified in the rules [2, 20]. These algorithms are not decision-tree based and with multiple dimensions require memory for backtracking search.

IP Routing lookup algorithm with good average case performance were considered by Cheung et al [6], which proposed a trie based scheme that takes into account frequencies in which prefixes are accessed, and Gupta et al [17], which proposed a decision tree based scheme with bounded worst-case and better average case performance. These average-case algorithms are specifically designed for the particular structure of prefix lookups, but their conclusions that traffic patterns are stable over time and average performance can be significantly improved by traffic-aware classifiers are consistent with ours.

3. DATA

Our evaluation is based on comprehensive data collected from a Tier 1 ISP. Packet classification is applied by edge routers to all traffic entering the network.

3.1 Filters

The filters are defined in the router configuration files. The configuration file of each router then associates each of the routers interfaces with a named filter. Often the same filter is used for multiple interfaces.

Each filter is specified in Extended ACL format as a list of rules. Each rule is a conjunction on several fields. A packet matches the rule if there is a match on all fields. Each rule has wildcarded bit masks on the address (source and destination) fields; a protocol specification (such as tcp, icmp, udp, or "ip" for all protocols) on the protocol field; and a range of values (range, equal, not equal, greater than, less than, or unconstrained) on the port field (Our data had only destination port specified). For the ICMP protocol, the destination port field specifies ICMP parameters. The rules also contain additional specifications (like TCP flags or fragments). The *domain* of a rule is defined as all packet values that match the rule. Each rule also has an *action*: a binary permit or deny on our data, but could also be more elaborate such as class of service. A packet can match several rules in the filter. The action of the filter on a packet is the action of the lowest-index rule (aka highest priority rule) that matches the packet. There is a default action that applies if none of the rules matches the packet. The default action is deny for deny/permit filters. Thus, the order of the rules is important for the semantics of the filter, which can change if rules with overlapping domains and different actions change their relative order. Table 1 shows a format of an 11-rule filter used in our data.

3.2 Traffic

The bulk of traffic in the network is captured by (sampled) netflow. Packets are filtered by edge routers and permitted packets are captured inside the network. Therefore, our flow records include only packets that are permitted at the edge routers and we study the performance of classification schemes with respect to permitted traffic only. We feel it is justified as permitted packets constitute the vast majority of packets.

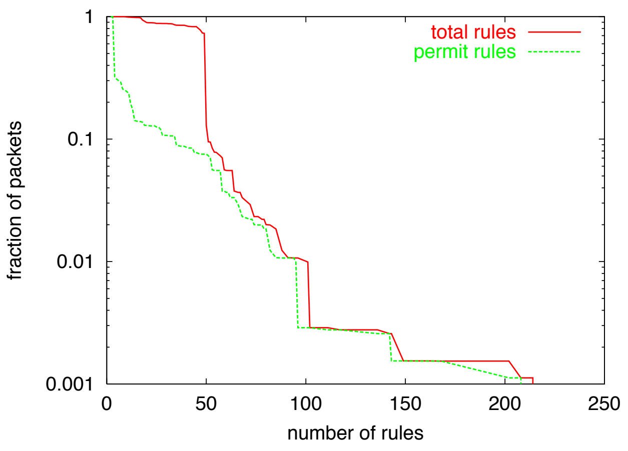


Figure 1: Distribution of filter size (weighted by respective packet traffic). Shows total number of rules and number of rules with a permit action.

Each sampled traffic record lists an associated ingress router (or a subset of possible ones) that is derived using the applicable configuration of the routing protocols. The methods used to derive the ingress interface from a flow record is described in [12]. The records have also been further sampled using size dependent sampling [9]. We note that the data was sampled with respect to bytes and we are interested in packets, but since number of packets and number of bytes in a flow are related in a close to proportional manner, it does not significantly influence our results.

The flow records were associated with over a hundred different ingress routers, the majority of them access routers and the other IGR (Internet Gateway Routers). Routers had between 1 to 175 interfaces. All together, there were close to 3600 interfaces that had packet data. There were about 3000 filter definitions (some filters are associated with several interfaces of the same router). 458 of the interfaces had at least 2000 daily records from sampled netflow, and we used these for our traffic-aware evaluation. These interfaces with at least 2000 daily records handled 99.4% of traffic. The filters associated with these interfaces contained up to 220 rules. (since we used interfaces with at least 2000 flow records, it follows from [9] that our standard error is at most 2%).

Figure 1 shows the distribution of filter sizes (the size of the filter is the number of rules), weighted by the packet traffic that passes through interfaces to which the filter is assigned. The figure shows the distribution for both the total number of rules in the filter and for the number of rules with a permit action.

Our performance results are aggregated over the following sets of interfaces:

- **ALL** : All interfaces with at least 2000 (sampled netflow) daily flow records.
- **AL60** : A subset of **ALL** that only includes interfaces with filters containing at least 60 rules (5.6% of **ALL** traffic went through these interfaces.)
- **AL100** : A subset of **ALL** (and of **AL60**) that only includes interfaces with filters containing at least 100 rules. (1.1% of **ALL** traffic went through these interfaces.)
- **S200** : The set of all filters defined in the router configura-

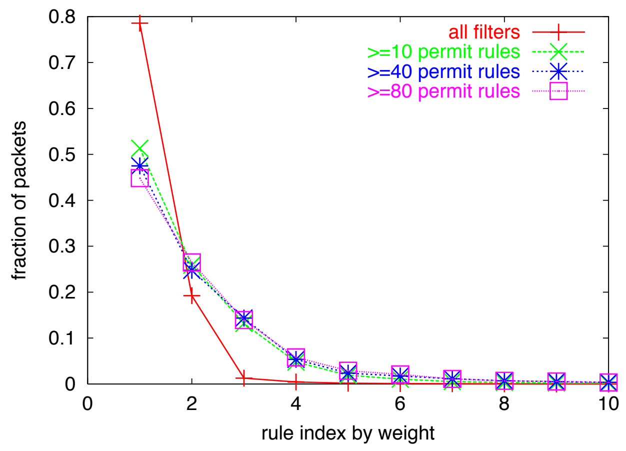


Figure 2: Distribution of rule usages, fraction of packets resolved by the ith heaviest rule. For ALL interfaces and broken down for interfaces with filters containing at least 10, 40, or 80 permit rules (these filters were applied to 25.2%, 8.7%, and 1.85% of traffic, respectively).

tion files with at least 200 rules. We obtained 78 such filters, including some filters with 1-2 thousand rules and filters that were not currently associated with any interface.

When aggregating results across **ALL** , **AL60** , and **AL100** sets, each interface is weighted by the amount of daily packet traffic that passed through it. Since **AL60** and **AL100** interfaces capture only a fraction of traffic, focusing on them allows us to see dependence of our results on filter size.

Since most **S200** filters have insufficient or no traffic data, when aggregating results we use a uniform weight for each filter. For **S200** filters we only evaluated the "static" performance metrics (memory and worst-case time) of different classification schemes.

We next consider the usage of rules in each interface. A rule is considered to resolve a packet if it is the highest priority rule that matches the packet; The ith heaviest rule is the rule that resolves the ith largest number of packets among all the rules in the filter. Figure 2 reveals a Zipf-like usage pattern of rules.[1]

The results are aggregated for **ALL** interfaces, but also are broken down according to the number of rules with a "permit" action in the filter (filters containing at least 10, 40, or 80 permit rules).[2] Since our traffic data includes only permitted traffic, this breakdown addresses a concern that filters with a larger number of total rules and few permit rules will artificially bias the rule-usage results. We observe that the Zipf-like pattern prevails for filters with a large number of permit rules: Even for filters with 80 or more permit rules, 45% of packets are resolved by the heaviest rule.

3.3 Rule Structure

We examined filters and rules according to which fields (among the 4 main fields: source IP address, destination IP address, protocol, and destination port) are *constrained*. We looked at all *patterns* (combination of constrained fields); we consider an address field constrained if at least one of the 32 bits is specified.

Table 2 shows the fraction of rules for each pattern. For **ALL** interfaces, the most common pattern by far was with the source address constrained and the other three fields wildcarded. Other patterns with at least $.01\%$ occurrences were protocol and source and destination addresses; only the two address fields; protocol, port and source address; and port and protocol. For **S200** filters, the most common pattern (71% of rules) had the source and destination fields constrained; and the second most common pattern (29%) had only the source address field constrained. Other two patterns (with about 0.05% occurrence) are the source, protocol, and port fields; and the protocol and port fields. The pattern where all fields are unconstrained corresponds to "permit all" rules (which are specified as the trailing rule in some filters in order to obtain a default permit action.) More rules have the address fields constrained since the address fields have a richer natural set of relevant values. The port field is essentially a refinement of the protocol field which typically is constrained only when the protocol is specified.

Table 3 shows pattern occurrence in filters: Each filter is counted as having a pattern if at least one of its rules has the pattern. Almost all filters have at least one rule with only the source address constrained; a rule with the source protocol and port fields constrained; and a rule with protocol and port fields constrained. The trailing permit all rule (that corresponds to the unconstrained pattern) occurs in smaller filters with fewer than 60 rules; these filters were applied to a significant fraction of traffic. Many filters contain a rule with both address fields constrained or with the destination address constrained.

				Interfaces			
src	dest	pr	po	**ALL**	**AL60**	**AL100**	**S200**
1	$*$	$*$	$*$	83.49	92.80	96.61	29.12
1	$*$	1	1	9.01	3.20	1.64	0.05
$*$	$*$	1	1	4.29	2.28	1.64	0.05
$*$	$*$	$*$	$*$	1.57	0.00	0.00	0.00
$*$	1	$*$	$*$	1.36	0.20	0.00	0.00
1	1	1	$*$	0.16	1.44	0.00	0.00
1	1	$*$	$*$	0.10	0.001	0.12	70.78
$*$	$*$	1	$*$	0.01	0.00	0.00	0.00
$*$	1	1	1	0.0001	0.00	0.00	0.00

Table 2: The percentage of use of each pattern. Patterns are specified with constrained fields having a "1" entry and wildcarded fields with a "$*$" entry. For ALL , AL60 , and AL100 , each rule is weighted by the fraction of packets subjected to its filter. For S200 , filters and rules are weighted uniformly.

				Interfaces			
src	dest	pr	po	**ALL**	**AL60**	**AL100**	**S200**
1	$*$	$*$	$*$	99.41	99.82	99.95	97.53
1	$*$	1	1	99.31	99.82	99.95	12.35
$*$	$*$	1	1	99.31	99.82	99.95	12.35
$*$	$*$	$*$	$*$	73.23	0.00	0.00	0.00
$*$	1	$*$	$*$	63.07	17.75	0.00	0.00
1	1	1	$*$	2.57	39.89	0.00	0.00
1	1	$*$	$*$	43.46	7.00	14.35	87.65
$*$	$*$	1	$*$	0.20	0.00	0.00	0.00
$*$	1	1	1	0.29	0.00	0.00	0.00

Table 3: The weighted percentage of filters with at least one rule with a certain pattern. For ALL , AL60 , and AL100 , each filter is weighted by the fraction of packets subjected to it. For S200 , filters are weighted uniformly.

Table 4 shows the amount of traffic directly resolved by rules

[1] The heaviest rules depend to an extent on the order of rules, which potentially can be changed without affecting filter semantics. We observed, however, that reordering does not change much this pattern.

[2] This breakdown is different than the **AL60** and **AL100** subsets that look at the total number of rules.

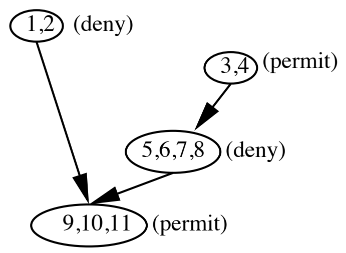


Figure 3: Partial order for the filter in Table 1

of certain pattern. The default rules (either the default "deny" or a trailing "permit all") resolve half the traffic in the smaller filters. Otherwise, rules with only the source field specified resolve the vast majority of packets. Other rules that resolve some traffic have the source, protocol and port fields specified, or the port and protocol fields specified.

				Interfaces		
src	dest	pr	po	**ALL**	**AL60**	**AL100**
1	*	*	*	52.92	97.45	99.68
*	*	*	*	40.73	0.00	0.00
default deny				5.76	1.98	0.00
1	*	1	1	0.40	0.58	0.32
*	*	1	1	0.18	0.00	0.00

Table 4: Fraction of traffic resolved by rules of certain structure. "Default deny" means that the packet did not match any rule and got resolved by the default (deny) action of the filter.

4. ALGORITHMS

We consider three basic performance metrics. The first is memory needed to store the data structure (we count the total number of rules and decision-nodes that are stored). The other two metrics are worst-case and average-case packet classification time (in terms of number of memory accesses to rules or decision nodes).

4.1 Decision lists

The simplest and most natural algorithm is to conduct a linear search through the ordered list of rules, until the first matching rule is found. We refer to such processing as a "decision list." We measure the memory and the worst-case time of a decision list by the total number of rules in the list. The average-case time for a decision list on a set of packets is the average, over packets, of the number of rules evaluated until the packet is matched. Although very efficient in memory, the linear worst-case processing time of decision lists is too slow for many settings [18].

We use decision lists as a reference point when looking at memory versus time tradeoffs. We also consider improving average-case time of a decision list by reordering rules (Note that memory and worst-case time remain the same under reordering, unless there is redundancy and rules can be removed without affecting semantics.)

Rules have a defined partial order on them. Any permutation that preserves that partial order also preserves the semantics of the filter: Two rules can change their order if they have the same action or if their domains are disjoint, otherwise, one precedes another according to their index in the decision list. The partial order on the 11-rule filter shown in Table 1 is illustrated in Figure 3.

The problem of reordering rules of a decision list to reduce average search time was studied in different contexts. Even in the special case where rules are commutative, the problem is known to be NP-hard; but the *greedy algorithm*, which iteratively selects and places the rule that resolves the largest number of remaining inputs (packets) has a small constant performance ratio[10, 8], and exhibits good performance on real-life data-sets for various applications [7, 4]. It is not hard to see that the worst-case performance of the greedy algorithm in the non-commutative case can be bad. We implemented *Extended Greedy*, which is an extension of greedy where each iteration performs a (locally optimal) placement of a rule along with all its predecessors (in the transitive closure). Observe that when rules are commutative (the partial order is flat), Extended greedy reduces to the plain greedy algorithm. The Extended Greedy heuristic does not have a bounded performance ratio on general instances: it is east to construct contrived instances on which it does not perform well. It is motivated, however, by particular observed properties of our filters data: One property is that our filter instances have long "commutative" stretches (In particular, rules that have the same action commute). Within each stretch, the greedy (and hence extended greedy) reordering are close to optimal. Another critical property is the Zipf-like rule usage which means that only the few heaviest rules really matter and moving them and their predecessors (that must be moved anyway) to the earliest possible spot, which is what extended greedy does, achieves close to optimal ordering. Observe that if there was a single rule that resolves all the traffic, then extended greedy would be optimal, since it will move this rule as far to front as possible. It is not hard to see that with Zipf-like rule usage distribution, the Extended Greedy algorithm has a small bounded performance ratio.

4.2 Decision trees

We construct decision trees in a top-down manner (as in previous work). We start with the root node that has all rules associated with it. At any point we consider a (currently a leaf) node and the set of rules associated with it. The decision whether to split the node and if so, which test (*cut-rule*) to use is *locally optimal* with respect to the optimization criteria and set of cut-rules: The cost of each cut-rule is computed under the assumption that the respective potential children are leaf nodes. The cost of not splitting the node is the cost of linearly searching the associated rules.

4.2.1 Cut-rules

The cut-rules we use are 1-dimension generalizations of the original rules. We include all prefixes (ranges) (as proposed in [15] and adapted by later schemes), and all single specified bits appearing in any rule in the source or destination address specifications (as suggested in[25]), all port ranges (being equal, less than, more than, or in the range of any specified port value[3]), and all protocols. Naturally, more generalizations can be considered like k-bit combinations of the source or destination addresses, and high degree multiple-fields cut-rules [21], but although it is always possible to obtain better trees by expanding the set of cut-rules, the set of all possible generalizations is exponential so we can not use them all. Since we use a rich set of possible cut-rules and the same set applied in the constructions of all types of decision trees, we believe that our conclusions on relative performance of the different schemes are robust to these variations.

4.2.2 Node splitting: standard and common-branches

When a node is split, the rules associated with it are assigned to its children, according to the relation between the domain of the rule to the domain of the selected cut-rule. The domain of the rule can be *contained* in the domain of the cut-rule, be *disjoint* to it, or

[3]Note that it is sufficient to consider port values that are stated in some rule

neither, in which case we say that the rule is *overlapping*. Packets that fall in the domain of the cut-rule can not be matched by rules with domains disjoint to that of the cut-rule, and vice versa. Thus, the domains of contained and disjoint rules are mutually exclusive.

Consider a node with a set of rules R and a cut-rule. Denote by C, D, and O the subsets of R with domains that are contained, disjoint, and overlapping the domain of the cut-rule. We consider two basic forms of branching: *standard splitting*, where rules with domains that overlap the cut-rule are replicated and associated with both decision branches (that is, one branch get rules $C \cup O$ and the other branch has rules $D \cup O$) and *common branches*, where overlapping rules are assigned to a third, *common branch* (That is, the common branch has the O rules and the two decision branches have rules C and D.) The common branch is explored by any packet that is evaluated at a node; by evaluating the cut-rule only one of the decision branches is explored. Note that the splitting is done per node, thus, it is possible to use both standard and common-branches splitting in a single decision tree. In our evaluation, however, we consider pure standard or common-branches decision trees.

Wildcards in rules can result in many overlapping rules that get replicated in both branches of standard decision trees. Replication can thus potentially increase the size of the tree by a constant factor on each level. In the worst-case, the size of a standard decision tree can be exponential in the size of the respective decision list. Common branches avoid replication of rules and thus result in trees that have worst-case size that is *linear* in that of the decision list.

With standard decision trees, we implemented a heuristic of *pushing common rules upward:* Rules that are common to all leafs that are descendents of a decision node are processed at the decision node (at the furthest ancestor that has the rule common to all its leaves.) This heuristic was proposed in [21] and led to significant reduction in storage. This heuristic is applied *after* the tree is computed, and as a result, interior nodes have rules associated with them. We refer to the resulting decision trees as *enhanced standard decision trees* and to the original decision trees without pushing rules upward as *plain standard decision trees*. Enhanced standard trees may seem reminiscent of common branching, as they avoids rules replication. The qualitative differences are that worst-case size of enhanced standard decision trees is exponential; pushing rules upwards is applied in retrospect after the tree is computed whereas common branching is integrated with the local optimization; and the rules resident (pushed upward to) an interior node are linearly searched.

An effective heuristic to control memory size in standard decision trees is to halt node splitting when the node has fewer than some set number of rules [21]. This heuristic can also be performed with common-branches trees and results in worst-case size that is at most a fraction larger than the original number of rules.

4.2.3 Evaluation procedure

The evaluation procedures for standard (plain or enhanced), and common-branches decision trees have acyclic flow-chart form: Both assume random memory access, rules are evaluated in order, and each cut-rule has associated "conditional jump statement." For standard decision tree, evaluation starts at the root and follows cut-rules down to a leaf node. Linear search is then performed on the rules present at the leaf node. For enhanced standard trees, rules present at interior nodes are linearly searched before the respective cut-rule is evaluated. Figure 4 shows an enhanced standard decision tree and the corresponding evaluation procedure. Sets of rules (that are pushed up to respective internal nodes or present at leaves) are denoted by capital letters.

Common-branches decision trees are evaluated such that (recursively) the common branch is explored first and after that the cut-rule is evaluated and the relevant decision branch is explored. Even though we provided a recursive definition to the order of evaluation, the evaluation procedure itself need not be recursive and has a simple flow-chart form. In particular, there is no need to maintain extra state as in a recursive evaluation. Figure 4 shows a common-branches decision tree and the respective evaluation procedure. Evaluation procedures for common-branches decision trees

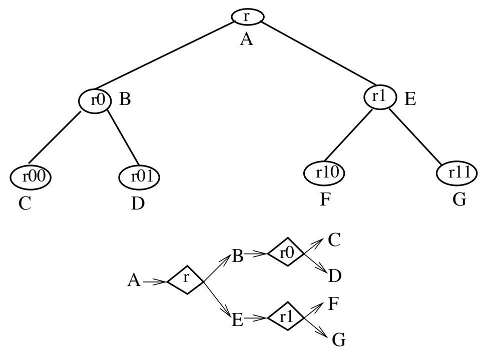


Figure 4: Standard decision tree and evaluation flow-chart (sets of rules are denoted by capital letters)

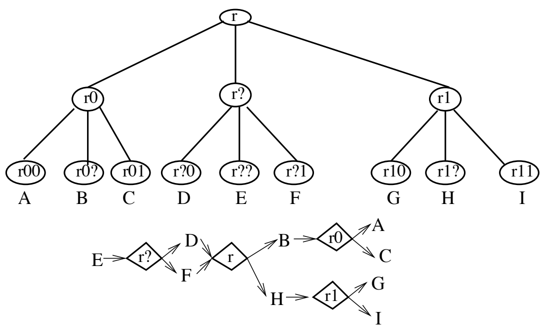


Figure 5: Common-branches decision tree and evaluation flow-chart (sets of rules are denoted by capital letters)

generally have a directed acyclic graph (DAG) format whereas those for standard decision trees always have a tree format. Like standard decision-trees, the evaluation of decision-trees with common branches can be fully pipelined.

We used the following refinements to speed up average-case time. We start our algorithm not with the original order of rules but with a reordered list that is computed using the Extended Greedy algorithm. The evaluation procedure has (ordered) sublists of rules (all rules associated with a leaf node or pushed upward rules at interior nodes) that are linearly searched. We maintain the index of the minimum index matching rule encountered so far, and the processing of each sublist is halted when that minimum index is exceeded.

Our performance metrics for decision trees generalize those of decision lists. The memory is measured by the total number of rules (including replicated rules) plus the number of interior nodes. The evaluation time of a packet is the total number of rules and cut-rules that are resolved until the action is determined. Each evaluation of a packet reaches a leaf of the decision tree after following decision branches on a path from the root. For standard decision trees, the processing time is at most the total number of rules and interior nodes (counting rules present at interior nodes and the leaf) on the path; but can be smaller with the refinements we implemented. For common branches decision trees, we also account for rules and cut-rules that are evaluated using the common branches.

The evaluation procedure for the standard decision tree in Figure 4 has size $3+|A|+|B|+|C|+|D|+|E|+|F|+|G|$ and worst-case time $2+|A|+\max\{|B|+\max\{|C|,|D|\},|E|+\max\{|F|,|G|\}\}$. A packet that "reaches" the leaf $r00$ has processing time at most $2+|A|+|B|+|C|$. The evaluation procedure for the common branches decision tree in Figure 5 has size $3+|A|+|B|+|C|+|D|+|E|+|F|+|G|+|H|+|I|$ and worst-case time $3+|E|+\max\{|D|,|F|\}+\max\{|B|+\max\{|A|,|C|\},|H|+\max\{|G|,|I|\}\}$. The processing time for a packet that "reaches" leaf $r00$ is at most $3+|E|+\max\{|D|,|F|\}+|B|+|A|$.

4.2.4 *Optimization criteria*

We construct decision trees under three different local optimization criteria: *worst-case*, *average-case*, and *mixed*.

Worst-case: The cost of not splitting the node is $|R|$ (linear search of the rules associated with the node). The cost of a split decision with a given cut-rule is $1+\max\{|C|+|O|,|D|+|O|\}$ for standard splitting ("1" for evaluating the cut-rule and the maximum number of rules between the two decision branches). The cost of a split with a common branch is $1+|O|+\max\{|C|,|D|\}$ (the cut-rule, with cost of 1, and the common branch, with cost $|O|$, are always evaluated, and we use the maximum cost of the two decision branches $|C|$ or $|D|$.) The split costs of each cut-rule are the same for standard and common-branch splitting, but since the number and content of branches is different we generally end up with different trees (unless the set O of overlapping rules is empty). If the minimum cost cut-rule has a lower cost than not splitting, the node is made an interior node and the appropriate rules are assigned to its branches. Otherwise, the node remains a leaf. The greedy splitting is then applied recursively to each branch.

Average-case (traffic aware): We consider all recorded packet traffic associated with the node. We first define the average-case linear-search cost for a set of packets on a node to be the average, over packets, of the number of rules (among the rules associated with the node) that is traversed until a matching rule is found or until the index of the current rule exceeds the index of the lowest-index matching rule seen so far for the respective packet (prior to visiting that node). The average-case no-split cost is simply the average-case linear-search cost of the node. The average-case split cost for a given cut-rule is as follows: For standard splitting it is 1 (the cut-rule is always evaluated) plus the sum over the two decision branches of the fraction of packets assigned to the branch times the average-case linear-search cost of these packets on the rules assigned to the branch ($C \cup O$ or $D \cup O$). For common-branch splitting, the average-case split cost is calculated as 1 (for cut-rule evaluation) plus the average-case linear-search cost of all packets on O rules plus the sum over the two decision branches of the fraction of packets going to the branch times the average-case linear-search cost of these packets on the rules of the branch (C or D).

Mixed: We used a weighted average of the worst-case and average-case split costs (and no-split cost) for each cut-rule. Our results are shown for a 50%-50% weighted average. Since the average-case cost is always smaller than worst-case cost, this results in a (local) decision that has worst-case cost that is at most twice that of the best worst-case cost.

5. COMMON BRANCHES VERSUS STANDARD SPLITTING

When wildcards are extensively used, standard decision trees can become significantly larger than the original decision list: If a cut-rule is specified on a single field (say, source address only), then all rules that are wildcarded on that field (for example, rules that only specify protocol or destination address) get replicated in all children. If (as in [21]) a cut-rule is specified on multiple fields, e.g. source and destination addresses, then rules that are specified on fields that include only a strict subset of these fields (e.g., source address only or destination address and protocol only) get replicated in multiple children. Cuts that result in multiple children can further exacerbate the size blowup.

Common branches decision trees have memory use that (even in the worst-case) is close to that of the original decision list and thus can be significantly smaller than standard decision trees. The evaluation time of standard and common branches trees depends on the rules and set of cut-rules but generally is longer for common branches trees (since the common branches need to be evaluated in addition to the decision branches). The actual performance gap between common branches and standard trees depends on the structure of the rule sets. We attempt to qualitatively illustrate this by considering three idealized rule set properties:

- At any point in the constructions, the best cut-rule has no overlapping rules. This can occur when rules are highly specified on a common area. Since there are no overlapping rules, there is no rule replication in standard trees and no rules are assigned to common branches. Therefore, standard and common branches constructions yield the same decision trees:
- Wildcards are *randomly and independently placed* in each rule. We refer to this as the "random property."
- The rule set is *structured* in that it contains several "independent" (disjoint) patterns (e.g., source only or port and protocol only or fractured packets and destination only or packet size only). We refer to this as the "structured property."

We explore the random and structured properties through analysis of simple rule sets.

Random wildcard placements. Consider a family of rule sets generated by setting each bit, in each rule, to one of $\{0,1,*\}$ uniformly and independently at random; and using 1-bit cut-rules. We obtain that the best cut-rule for any subset of rules has (by expectation) $1/3$ of the rules overlapping, $1/3$ disjoint, and $1/3$ contained. A standard decision tree has depth (and thus time) $\log_{3/2} R$[4] (the number of rules assigned to a node decreases by a factor of $3/2$ in each level) and size $R(4/3)^{\log_{3/2} R} = R^{1+\log_{4/3}(3/2)} \approx R^{1.71}$ (the total number of rules assigned to nodes in a level grows by a factor of $4/3$ with each level). A common-branches decision tree obtained for such a rule set has depth $\log_3 R$ (the total number of rules assigned to a node decreases by a factor of 3 with each level), size $1.5R$, and time $2R^{\log_3 2} \approx 2R^{0.63}$ (the time dependence is captured by the recursive relation $T(R) = 1 + 2T(R/3)$). The respective performance is summarized in Table 5. For "random property" rule-sets, common-branches decision trees use considerably smaller memory size but also considerably higher processing times than standard trees.

Structured wildcard placements. Consider filters with l rules over k fields, where each rule has one of the k fields completely specified, and the other $k-1$ fields wildcarded. Suppose that l/k rules are specified in each field and that subsets of rules that are

[4]We omit lower order terms from memory and time bounds.

	"random" property	
	standard	common branches
memory	$R^{1.71}$	$1.5R$
time	$\log_{3/2} R$	$2R^{0.63}$
	"structured" property	
	standard	common branches
memory	$l^{\Theta(k)}$	$\Theta(l)$
time	$\Theta(k \log(l/k))$	$\Theta(k \log(l/k))$

Table 5: Performance of standard and common branches decision trees for the two rule-set properties.

specified on a common field can be split using a balanced decision tree of logarithmic depth.

With this structure, cut-rules that are specified on more than one field are not effective, since all rules will be overlapping with respect to such a cut-rule. Thus the only effective cut-rules are those specified on a single field. For such a cut-rule, all rules that are not specified on the same field as the cut-rule will be overlapping.

Consider standard decision trees for such rule-sets and the processing of a node with a subset of the original rules at a node. The most effective cut-rule can only split half of the rules in a certain dimension. Thus, the depth of the tree (and the worst-case time) is $\Theta(k \log(l/k))$. Rules that are overlapping get repeatedly replicated under both branches. The total size of the resulting tree is $l^{\Theta(k)}$.

We next consider decision trees with common branches. At each decision node we select a cut-rule that best splits rules associated with one field, rule that are not specified on that field are assigned to the common branch. The resulting tree has the structure depicted in Figure 6, of a single common-branches path with hanging sub-trees that each includes all rules specified on a certain field. The depth of this tree is $\Theta(k)$, its size $\Theta(l)$, and the worst-case search time $\Theta(k \log(l/k))$. (See Table 5 for a summary). It follows that for such rule-sets common-branches decision trees have similar worst-case time and considerably better memory than standard decision trees.

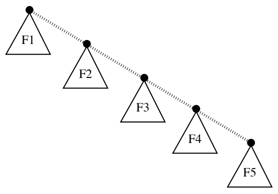


Figure 6: Common branches for wildcarded rule sets. Common branches are marked by dotted lines. Each triangle is a decision tree applied to all (and only to) rules specified on that field. The resulting evaluation order is a sequential field-by-field evaluation.

These rule-sets properties show that the respective performance of standard versus common branches trees highly depends on the wildcards structure. Without wildcards, common branches degenerate to standard decision trees; there are no overlapping rules and thus there is no replication by standard trees or rule assignment to common branches. When wildcard positions are random, standard and common branches exhibit different memory-time tradeoffs, common branches use much smaller memory and standard trees have faster evaluation times. When strong correlations are present, common branches provide superior tradeoffs.

Although real-life rule sets do not completely fall in one category, we believe that the structured property captures enough of real-life rule sets to benefit from common branches. Note that common-branches decision tree construction does not require structure to be explicit: the "fields" need not be explicit, fully specified, or perfectly disjoint.

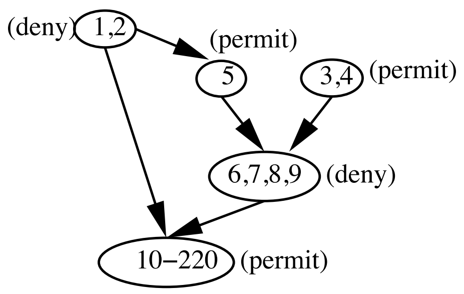


Figure 7: Partial order for a 220-rule filter

6. PERFORMANCE EVALUATION

We use our real-life data of filters and corresponding traffic to study the memory, worst-case time and average-time performance of the different classification algorithms. In particular, understand memory versus time tradeoff of the different decision trees and compare standard and common branches decision trees. We also quantify average-time versus worst-case time performance.

6.1 List reordering

Before presenting aggregated results, we examine the performance of Extended Greedy reordering on two interfaces.

Our traffic data for the 11-rule filter of Table 1 on an interface it was assigned to had all captured packets resolved by rule #9 of the filter. Since rule #9 must follow all preceding rules (see the partial order shown in Figure 3), the list can not be reordered such that rule #9 is placed earlier. Thus, in this particular case, reordering can not improve the average-case search time.

Figure 7 shows the partial order of another filter with 220 rules (rules are not listed). 70% of permitted packets were resolved by rule #82, about 30% by rule #27, and about 0.1% by rule #3. The Extended Greedy reordering algorithm moved rule #3 to the top of the list, rule #82 to position 10 and rule #27 to position 11. The average time was consequently reduced from 65.3 to 10.3.

This 220-rule filter was applied at several interfaces of the router. At another interface, 46% of packets got resolved by rule #66, about 47% by rule #21, 5% by rule #32, and the others (less than 1%) by rules #3 and #26. The Extended Greedy reordering algorithm placed rule #3 as the first one and rules #21, #66, and #32, and #26, in this order, in positions 10-13. The average-time consequently reduced from 42.3 to 10.6.

Observe that the variance in processing time is very low, since there were no observed flow that utilized other rules. This seems to be the typical situation (as depicted in Figure 2.)

Interestingly, two different orderings worked best for the two interfaces that utilized the same filter. This is because different interfaces serve packets from different source addresses. This is consistent with our observation that some rules never get utilized: Since filters tend to be designed according to general guidelines, often the same filter is used in multiple interfaces. Therefore, for any particular interface there could be rules that never apply to packets routed through the interface. This also suggests that traffic-aware optimization should be performed per interface and that performance can be sensitive to routing changes, as they can affect the traffic mix (for example, destination addresses seen).

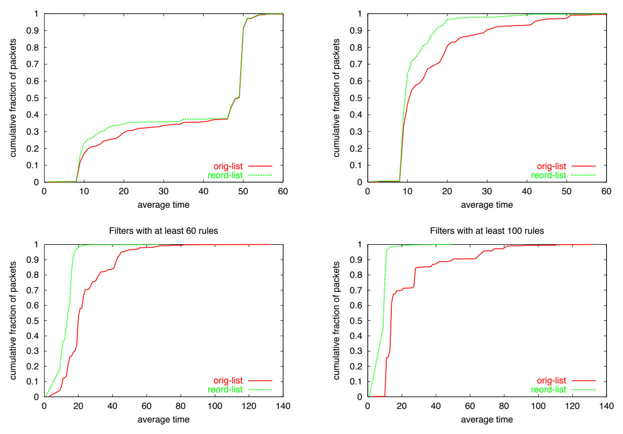


Figure 8: Average time for original and reordered decision list on ALL (top left), subset of ALL that only includes interfaces with filters without a "permit all" rule (top right), AL60 (bottom left) and AL100 (bottom right).

We next consider aggregated results on average-case time performance of original (ORIG-LIST) versus reordered by Extended Greedy (REORD-LIST) decision lists. ACLs have a default deny action where packets that are not resolved by any rule in the filter are denied. Our data had large fraction of filters (and associated traffic) that had a "permit all" as the last rule of the filter (thus making the filter a "default permit" action). On these "default permit" filters (all had fewer than 60 rules) it turns out that over 99% of packets were resolved by this last rule. The "permit all" rule is preceded by all other rules in the partial order and can not be moved up to a higher priority spot (unless there are redundant rules with permit actions that can be eliminated altogether), thus, the average time could not be improved by reordering rules on these filters. Figure 8 (top part) shows the average time distribution with or without reordering over all **ALL** , and also when restricted to the subset of interfaces with filters that do not have a trailing "permit all" rule. Figure 8 (bottom part) shows the distribution for **AL60** and **AL100** interfaces. We see that reordering can be very effective in decreasing average time for the restricted subset of filters without a "permit all" rule and for **AL60** and **AL100** interfaces (none of the filters with 60 or more rules had the trailing "permit all" rule.)

The average average-time performance is listed in Table 8 (**ALL** interfaces), Table 9 (**AL60** interfaces) and Table 10 (**AL100** interfaces). Recall that the memory and worst-case time are not affected by reordering.

6.2 Decision trees

Before presenting the aggregated performance metrics we provide a closeup view of the two interfaces with the 11-rule and 220-rule filters. These individual interfaces illustrate some general issues in the structure of different decision trees. Decision trees obtained using the different local optimization criteria for the 11-rule filter in Table 1 are shown in Figure 9 (standard decision trees) and Figure 10 (decision trees with common branches). The decision trees with common branches are shown together with the respective evaluation procedures. The performance of the different algorithms on our 11-rule filter is listed in Table 6.2 and on the 220-rule filter is listed in Table 6.2.

Recall that the packet traffic for the 11-rule filter had all packets resolved by rule #9. Therefore, for better average-case time, rule #9 is tested earlier (and its predecessors excluded). When optimizing

method	memory	wc	ave
decision list	11	11	9
standard decision trees			
worst case	26	7	5
+ pu	21	7	5
pure traffic aware	17	10	3
+ pu	15	10	3
mixed	26	8	4
+ pu	22	8	4
common branches decision trees			
worst case	14	7	6
pure traffic aware	13	11	3
mixed	13	9	4

Table 6: Performance on the filter in Table 1.

for worst-case, rule #9 can be placed later in the evaluation order.

The average-case optimized decision trees shown for the 11-rule filter (in Figures 9 and 10) show heavily populated leaf nodes. The reason is that these rules have zero-usage and further splitting of these nodes can not improve average case performance. These populated leafs increase worst-case times (since leaf nodes are searched linearly) but also improve memory usage (splitting adds interior notes and for standard trees has an even higher memory penalty due to rule replication).

Figure 10 shows an interior node with a single nonempty child branch. This can occur under average-case optimization. When a cut-rule can isolate all rules from a large enough fraction of packet traffic; after testing the cut-rule, the isolated packets get mapped to the default action and the rest of the packets are directed to the child. This can not happen under worst-case time optimization.

We also observe that the enhanced version of standard decision trees seems to use considerably less memory than the plain standard version; and the average-time performance under mixed optimization is very close to that of the worst-case optimization since the 50%-50% weighting biases towards the worst-case (that has larger average time).

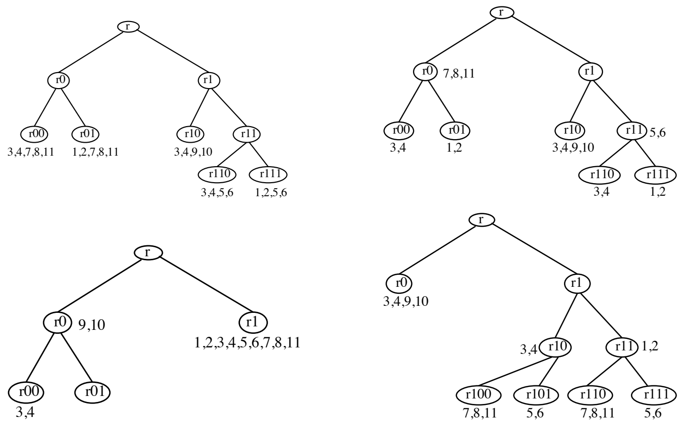


Figure 9: Standard decision trees for the filter in Table 1. Top: worst-case (left) and its enhanced version (right). Bottom: (enhanced versions of) pure traffic-aware (left) and mixed (right).

Aggregated results

A summary of the average performance of the different algorithms is shown in Table 8 (**ALL** interfaces), Table 9 (**AL60** interfaces), and Table 10 (**AL100** interfaces). The entries in each row of the tables are sorted by magnitude. The decision lists are denoted

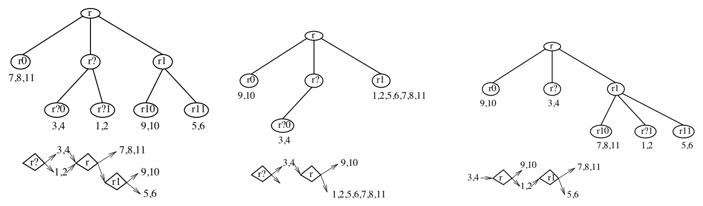


Figure 10: Common-branch decision trees for the filter in Table 1 (common branches are labeled with "?"): Worst-case (left), pure traffic-aware (middle), and mixed (right). Below each decision tree we show the respective evaluation flow chart.

method	memory	wc	ave
Decision lists			
original	220	220.0	65.6
reordered	220	220.0	10.3
standard decision trees			
worst case	512	12.0	9.0
+ pu	346	12.0	9.0
pure traffic aware	427	207.0	4.3
+ pu	224	207.0	4.3
mixed	546	12.0	9.0
+ pu	381	12.0	9.0
common branches decision trees			
worst case	298	13.0	9.6
pure traffic aware	222	207.0	4.3
mixed	299	17	9.3

Table 7: Performance on the 220-rule filter.

by ORIG-LIST (original order) and REORD-LIST (reordered). Decision trees are produced according to the three optimization criteria ("wc-" for worst-case, "ta-" for average-case (traffic aware), and "mx-" for mixed). Standard decision trees are denoted by "DT" for the plain variant and "DT+" for the enhanced variant. Common branch decision trees are denoted by "CBDT."

The tables show consistent patterns over the three sets of interfaces: decision lists are the most memory efficient; common-branches use much less memory than standard decision trees; enhanced standard decision trees use significantly less memory than the plain variant, but still more than common-branches. The tables also show that when optimizing for worst-case we use more memory than under pure traffic-aware optimization; which is explained by many rules with zero usage. Mixed optimization results in similar memory usage as worst-case optimization.

Memory. A more refined view of the aggregated results on memory usage is provided by the cumulative distributions shown in Figure 11. The left hand side figure includes standard decision trees (plain or enhanced), and shows that pushing common rules upward is a very effective heuristic. The right hand side figure shows common-branches and enhanced standard schemes and show a clear advantage for common branches. Common-branches trees use memory that is fairly close to that of decision lists. Consistent patterns were obtained for **AL60** and **AL100** interfaces.

When examining the ratio of size to decision list size for common branches, standard, and enhanced standard schemes optimized for worst-case time, we noticed different outliers "worst-case" behavior: The largest ratio for common branches trees was only 1.43 whereas the largest ratios for standard decision trees were 5 for enhanced trees and 9 for plain trees.

Worst-case time. Cumulative distribution for worst-case time for **ALL** interfaces is shown in figure 12 (consistent results were obtained for **AL60** and **AL100** interfaces). Decision lists, as expected, have the worst time. The worst-case time on standard and common-branches decision trees that have the same optimization criteria is very close. The pure-traffic aware optimization results in decision trees with considerably worse worst-case time than mixed or worst-case optimizations. The best performers, bunched together, are the worse-case and mixed optimized, common-branches and enhanced standard decision trees. (Plain standard decision trees have the same worst-case time as enhanced standard decision trees and are not shown in the figure.)

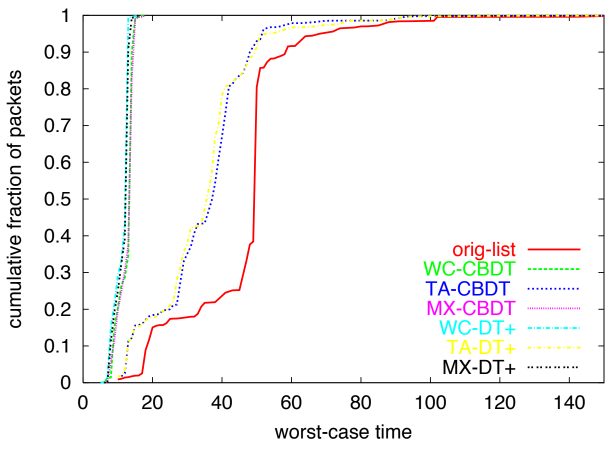


Figure 12: Worst-case time for ALL interfaces.

Average-case time. Cumulative average-time distribution for decision tree algorithms on **ALL** interfaces is shown in Figure 13. (Consistent results were obtained for **AL60** and **AL100** interfaces.) The best performers are the standard and common-branches schemes that are optimized for average-case time. The mixed and worst-case optimization yield decision trees with worse worst-case time. The average time performance of enhanced standard and common branches trees that are constructed under the same local optimization criteria is very close.

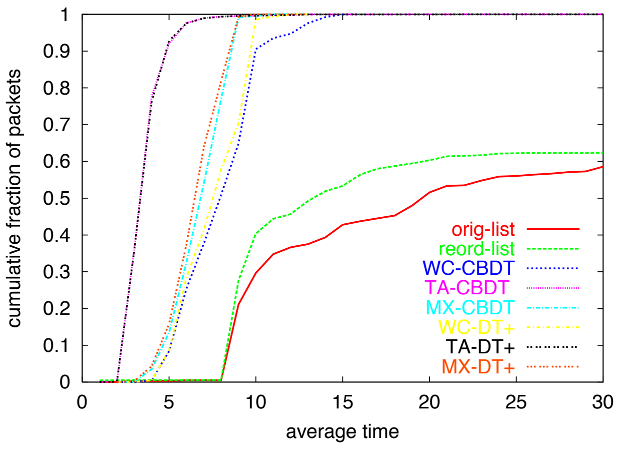


Figure 13: Average time (ALL interfaces).

Memory	ORIG-LIST 46.30	REORD-LIST 46.30	TA-CBDT 55.71	WC-CBDT 60.31	MX-CBDT 62.17	TA-DT+ 79.31	WC-DT+ 83.05	MX-DT+ 86.29	TA-DT 113.63	WC-DT 132.86	MX-DT 143.56
Worst-case	WC-DT 11.58	WC-DT+ 11.58	MX-DT 11.83	MX-DT+ 11.83	WC-CBDT 12.83	MX-CBDT 12.95	TA-DT 34.62	TA-DT+ 34.62	TA-CBDT 35.08	ORIG-LIST 46.30	REORD-LIST 46.30
Average	TA-DT 4.80	TA-DT+ 4.81	TA-CBDT 4.84	MX-DT 8.08	MX-DT+ 8.09	MX-CBDT 8.26	WC-DT 9.24	WC-DT+ 9.24	WC-CBDT 9.57	REORD-LIST 35.68	ORIG-LIST 37.10

Table 8: Memory, worst-case time, and average time, for ALL interfaces.

Memory	ORIG-LIST 82.05	REORD-LIST 82.05	TA-CBDT 84.58	TA-DT+ 104.37	MX-CBDT 107.61	WC-CBDT 111.71	WC-DT+ 141.96	MX-DT+ 150.17	TA-DT 152.63	MX-DT 207.77	WC-DT 212.85
Worst-case	WC-DT 11.59	WC-DT+ 11.59	MX-DT 11.64	MX-DT+ 11.64	MX-CBDT 13.01	WC-CBDT 13.29	TA-CBDT 59.62	TA-DT 65.68	TA-DT+ 65.68	ORIG-LIST 82.05	REORD-LIST 82.05
Average	TA-DT 4.42	TA-CBDT 4.43	TA-DT+ 4.44	MX-DT 7.55	MX-DT+ 7.56	MX-CBDT 7.81	WC-DT 8.31	WC-DT+ 8.31	WC-CBDT 8.57	REORD-LIST 13.51	ORIG-LIST 24.33

Table 9: Memory, worst-case time, and average time, for AL60 interfaces.

Performance on **S200** *filters.* On **S200** filters we observed that the worst-case time performance is similar for standard and common-branches decision trees, and both significantly outperform decision lists (linear search). Memory size is minimized by decision lists, common-branches trees are the next best performer, followed by the enhanced standard decision trees, and then by standard decision trees. These results are consistent with what we observed for **ALL** , **AL60** , and **AL100** interfaces: Common-branches decision trees emerge as the best performers, with similar worst-case time and significantly less memory than (enhanced or plain) standard decision trees. We also observe that enhanced standard decision trees use significantly less memory that plain standard decision trees.

Time versus memory. When examining outliers in the relative memory and time performance of standard and common-branches decision trees we observed that while the memory ratio can be large, the worst-case time ratio is small and bounded. On **ALL** interfaces, common branches decision trees were never more than 50% slower than standard trees. Interestingly, however, the outliers with large memory gaps are not those with large time gaps This suggests that it is worth while to construct both types of trees and choose one with better tradeoffs. We can always select a scheme with worst-case time that is within 1.25 factor and memory that is within a 1.5 factor from that of the best one. This suggests that hybrid trees of plain and common branches nodes can be of interest.

Performance stability. Our traffic-aware decision trees were constructed using full day traffic data (August 29, 2004). When applying them to traffic data collected on the same interfaces in the subsequent three weeks we observed that average-case performance remains virtually unchanged.

7. CONCLUSIONS

We propose common-branch decision trees for packet classification. Common-branches trees use linear amount of memory in the worst-case as they avoid rule replication that occurs with standard decision trees. They do have the benefits of standard decision trees classifiers including a very simple packet evaluation procedure that can be fully pipelined. Our experimental evaluation on real-word filters showed that common branches trees use much less memory than standard decision trees and have comparable worst-case and average-case search times. We attribute the good performance of common branching to presence of extensive wildcarding with certain structure in the rule sets.

A forward looking question is whether this performance gain of common branches will prevail as filters evolve. Correlations and structure of this type occurs in many different domains of real-life, from Web access patterns to word occurrence in documents to market basket data and thus are likely to persist. While our real-life rule-sets used 4 main fields with one or two dominant patterns per filter, filters are evolving to be larger and more diverse (with more patterns being used over a larger number of fields). Even with large number of fields, we still expect a small number of fields to be specified in any one rule. These trends will lead to an increase in the performance gain of common branches.

We suggest that average-case time is an important metric in some packet classification settings and propose algorithms with improved average case performance. We show that these algorithms are very effective on our data; and qualitatively explain it by the Zipf-like pattern of rule usage in the filters. Can we expect this pattern to prevail as filters and rules get more complex? We believe that it will: Zipf-like patterns are observed in many Internet applications and realms of life, and it seems likely that we will find them under emerging applications of packet classifications: For QoS filters, we can expect that the bulk of packets fall in some dominant classes of service; when an interface serves multiple VPNs with different classification requirements, we can expect that the capacity usage of different customer VPNs is Zipf-like.

We discuss some avenues for further study. While we constructed decision trees with all decision nodes being standard or common-branches, one can construct hybrid trees, with some interior nodes being standard decision nodes and others having common branches. In fact, our code handles general hybrid trees, which like pure common branches have an evaluation procedure with a DAG form (the only difference is that rule replication occurs and the same rule can appear in multiple places.). Likewise, we constructed trees where the local optimization criteria was geared to either the average-case metric, worst-case metric, or a mixed 50%-50% weighting of the two. The 50%-50% mix resulted in performance that is close to that of the worst-case. It can be interesting to explore the full spectrum of tradeoffs and in particular see to what extent we can retain average-case performance while bounding the worst-case performance. Last, while our evaluation utilized cut-rules with binary and single-dimension branching, common branches can be used with more complex cut-rules including non-binary[15] branching (where each node has multiple children, each corresponding to a different range of the same dimension) or multi-dimensional cuts [21] (where hypercubes are used instead of ranges).

Acknowledgement. We thank Lee Breslau, Balachander Krishnamurthy, and Jia Wang for discussions and suggestions.

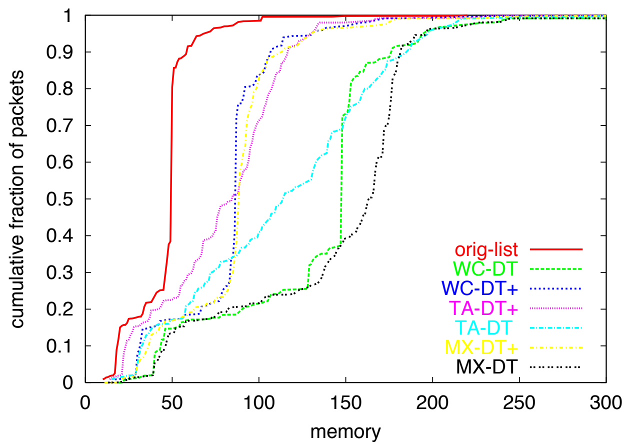
8. REFERENCES

[1] Controlling network access with access control lists, 2004. http://cisco.com/univercd/cc/td/doc/product/lan/cat6000/mod_icn/fwsm/fwsm_2_2/fwsm_cfg/mngacl.pdf

[2] F. Baboescu, S. Singh, and G Varghese. Packet classification

Memory	ORIG-LIST 123.63	REORD-LIST 123.63	TA-CBDT 125.96	TA-DT+ 138.72	MX-CBDT 163.88	WC-CBDT 167.02	WC-DT+ 203.54	MX-DT+ 211	TA-DT 231.97	MX-DT 291.82	WC-DT 293.29
Worst-case	WC-DT 11.14	WC-DT+ 11.14	MX-DT 11.53	MX-DT+ 11.53	WC-CBDT 13.07	MX-CBDT 13.08	TA-CBDT 100.56	TA-DT 101.31	TA-DT+ 101.31	ORIG-LIST 123.63	REORD-LIST 123.63
Average	TA-DT 3.81	TA-CBDT 3.81	TA-DT+ 3.81	MX-DT 7.92	MX-DT+ 7.92	MX-CBDT 8.15	WC-DT 8.61	WC-DT+ 8.61	WC-CBDT 9.24	REORD-LIST 9.98	ORIG-LIST 22.58

Table 10: Memory, worst-case time, and average time, for AL100 interfaces.



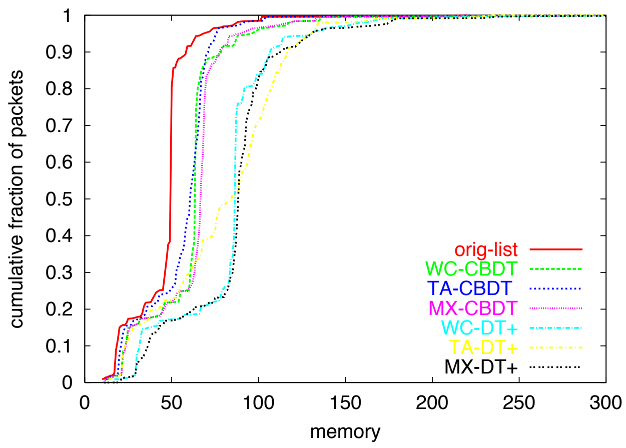


Figure 11: Memory for enhanced and plain standard decision trees (left) and common-branches and enhanced standard decision trees (right).

for core routers: Is there an alternative to cams? In *Proc. of IEEE Infocom 2003*, 2003.

[3] F. Baboescu and G. Varghese. Scalable packet classification. In *Proc. of ACM SIGCOMM 2001*, 2001.

[4] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. Adaptive ordering of pipelined stream filters. In *SIGMOD*. ACM, 2004.

[5] A. Brodnik, S. Carlsson, M. Degermark, and S. Pink. Small forwarding tables for fast ip lookups. In *Proc. of ACM SIGCOMM 1997*, pages 3–13, 1997.

[6] G. Cheung and S. McCanne. Optimal routing table design for IP address lookups under memory constraints. In *Proc. of IEEE Infocom 1999*, 1999.

[7] E. Cohen, A. Fiat, and H. Kaplan. Associative search in Peer to Peer networks: Harnessing latent semantics. In *Proceedings of the IEEE INFOCOM'03 Conference*, 2003.

[8] E. Cohen, A. Fiat, and H. Kaplan. Efficient sequences of trials. In *Proc. 14th ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[9] N.G. Duffield, C. Lund, and M. Thorup. Learn more, sample less: control of volume and variance in network measurements. *IEEE Transactions on Information Theory*. to appear.

[10] U. Feige, L. Lovasz, and P. Tetali. Approximating min-sum set cover. In *Proceedings of 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, volume 2462 of *LLNCS*, pages 94–107. Springer, 2002.

[11] A. Feldmann and S. Muthukrishnan. Tradeoffs for packet classification. In *Proc. of IEEE Infocom 2000*, 2000.

[12] Anja Feldmann, Albert G. Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, and Fred True. Deriving traffic demands for operational IP networks: methodology and experience. In *SIGCOMM*, pages 257–270, 2000.

[13] P. Gupta, S. Lin, and N. McKeown. Routing lookups in hardware at memory access speeds. In *Proc. of IEEE Infocom 1998*, 1998.

[14] P. Gupta and N. McKeown. Packet classification on multiple fields. In *Proc. of ACM SIGCOMM 1999*, 1999.

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

[16] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15, 2001.

[17] P. Gupta, B. Prabhakar, and S. Boyd. Near optimal routing lookups with bounded worst-case performance. In *Proc. of IEEE Infocom 2000*, 2000.

[18] T. V. Lakshman and D. Stiliadis. High speed policy based packet forwarding using efficient multi-dimensional range matching. In *Proc. of ACM SIGCOMM 1998*, 1998.

[19] J. F. McMullen. Get secure with Cisco extended IP access control lists, 2001. http://techrepublic.com.com/5102-6265-1058307.html.

[20] L. Qiu, G. Varghese, and S. Suri. Fast firewall implementation for software and hardware based routers. In *Proc. of ICNP 2001*, 2001.

[21] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *Proc. of ACM SIGCOMM 2003*, 2003.

[22] V. Srinivasan, S. Suri, and G. Varghese. Packet classification using tuple space search. In *Proc. of ACM SIGCOMM 1999*, 1999.

[23] V. Srinivasan and G. Varghese. Faster IP lookups using controlled prefix expansion. In *Proc. of ACM Sigmetrics 1998*, 1998.

[24] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. In *Proc. of ACM SIGCOMM 1998*, 1998.

[25] T. Woo. A modular approach for packet classfication: algorithms and results. In *Proc. of IEEE Infocom 2000*, 2000.