# Rules in Play: On the Complexity of Routing Tables and Firewalls

H. B. Acharya, Satyam Kumar, Mohit Wadhwa, and Ayush Shah
IIIT Delhi
Email: acharya@iiitd.ac.in

***Abstract*—Networking infrastructure, such as routers and firewalls, consist of a policy (i.e., where to forward which packets) and a mechanism that implements it. As the correctness of the policy is critical, it is a natural candidate for formal verification. Indeed, several verification algorithms have been developed, that detect anomalies, conflicts, and redundancies in practical firewalls and flow tables. However, theory suggests that the problem is intractable in general: the decision tree for a policy is of size $O((2n)^d)$, where $n$ is the number of rules and $d$ is the number of observed features used in making the decision. (In a typical firewall, $n = 1000$ and $d = 10$.)**

**In this paper, we show why the verification of practical firewalls is not as hard as previously thought. Using a new concept, "rules in play," we find a new, tight bound on the size of the decision tree, and suggest three other factors - narrow fields, singletons, and all-matches - that make the problem tractable in practice. We also present an algorithm to solve an open problem: pruning a policy to the minimum possible number of rules, without changing its meaning.**

## I. Introduction

In packet-switched networks, packets are routed or filtered at each hop according to the policy in a router or middlebox. (Such policies are called flow tables, routing tables, firewall rulesets, etc.) As the rules in a policy can interfere with each other, the complexity of a policy is not simply described by the number of rules. We propose to measure the complexity of a policy by the size complexity of its decision tree.

How complex is a policy in general, and how should it be stored for fast processing of packets? This is an important and well studied question; several groups have proposed optimized representations for policies, using tries [1], lookup tables [2], B-trees [3], or skip lists [4]. However, the question remains whether it is practical to use such representations, or real policies are too complex. Following Liu [5], we know that the theoretical worst-case complexity of a policy (measured by the size complexity of its decision tree) is $(2n-1)^d$, where $n$ is the number of rules and $d$ is the number of fields, i.e. features of interest. In other words, for a typical policy, where $n$ is several thousand and $d$ is $5-10$, the decision tree is intractably large (in the worst case); no accurate representation of the policy can be both small and fast. But in sharp contradiction to this negative result, other groups have successfully built decision trees, and used them to analyze policies for entire networks [6]!

The natural question that follows is, are the algorithms in fact intractable (and we have just been lucky in practical experiments), or is the theory overly conservative? The first contribution of this paper is a closer look at this question. By a minimal extension to decision diagrams, to keep track of "rules in play", we refine the known upper bound on decision diagram size: the decision diagram for a policy of $n$ rules and $d$ fields is tightly upper bounded by the Delannoy number $D(n-1, d)$. We also suggest how the mathematical modeling of policies can be improved, by taking into account some new metrics of policy complexity. Further, we indicate that "friendly" values of these metrics may explain why decision-diagram algorithms have proven tractable in practice.

Our first contribution is, therefore, to the theoretical analysis of policies - the study of anomalies [7], inter-rule conflicts [8], redundancies, and so on. For example, Frantzen [9] provides a framework for understanding the vulnerabilities in a firewall, and Blowtorch [10] is a framework to generate packets for testing. These algorithms depend on the complexity of a policy, as measured by the size of its decision diagram; our work suggests why they are practical to use.

Our study of extended decision diagrams has also produced an unexpected side benefit: the first algorithm to prune redundant rules and produce a true *minimum* policy, i.e. the smallest policy with the same semantics as the original. Current state of the art algorithms [11] [12] only produce *minimal* policies, by trimming redundant rules until no more can be found. In practice, high-throughput systems, such as backbone routers, use special hardware - ternary content addressable memory [12], pipelining systems [13], etc. - which are not only expensive, but limited in the size of policies they can accommodate. Therefore, there is active research on algorithms to shrink policies [12]; if our algorithm can be scaled to large policies, it will also be a contribution to the use of accelerators in packet resolution.

We begin by providing our definitions and concepts in the next section, and also discuss our new metrics, the decision diagram construction algorithm, and rules in play. After this, we devote one section each to show a tight bound on the size of decision diagrams, and to demonstrate how mitigating factors - narrow fields, singletons, and all-matches - can reduce the size of decision diagrams in practice. We then present our redundancy removal algorithm, LeafTrim, and our experimental results. We end with a few concluding remarks.

## II. Terms and Concepts

In this section, we define the terms and concepts used in the paper, such as policies and properties; our new metrics,

978-1-5090-3281-5/16/$31.00 ©2016 IEEE

*oneprob*, *allprob*, and *fieldwidth*; and discuss decision diagrams and "rules in play".

### A. Packets, Rules, and Matching

In our work, we model a *packet* as a $d$-tuple of non-negative integers. Why this model? In order to decide what to do with a packet (whether to forward it, which interface to forward it on, etc.), routers and firewalls examine its various attributes - usually values in the packet header, such as source address, destination address, source port, destination port, protocol, and so on. (In 'deep packet inspection', attributes from the packet payload are also checked.) The $d$ fields of our packet model represent the $d$ features examined.

A *rule* represents a single rule in a flow table. It consists of two parts: a *predicate* and a *decision*.

The rule predicate is of the form

$$x_1 \in [x_{1,1}, x_{1,2}] \wedge x_2 \in [x_{2,1}, x_{2,2}] \ldots x_d \in [x_{d,1}, x_{d,2}]$$

where each interval $[x_{k,1}, x_{k,2}]$ is an interval of non negative integers, drawn from the domain of field $k$. (For example, suppose the third field in packets and rules represents source IP address. In IPv4, the domain of this field is $[0, 2^{32} - 1]$. Then, in any rule, $0 \leq x_{3,1} \leq x_{3,2} \leq 2^{32} - 1$.)

The decision is an action, such as (in a firewall) *accept* or *discard*. [We call a rule with decision *accept* an "$accept$ rule", and a rule with decision *discard* a "$discard$ rule".]

A packet that satisfies the predicate of a rule is said to *match* the rule. For example, the packet $(1, 26, 7)$ clearly matches the rule

$$x_1 \in [0, 108] \wedge x_2 \in [21, 65535] \wedge x_3 \in [7, 616] \rightarrow accept$$

### B. Policies and Packet Resolution

A *policy* consists of multiple rules, as defined above, and a specification of precedence - i.e. some way to decide which action to execute, if multiple rules match a given packet. There are two methods in use to decide the precedence of matching rules.

1) *First Match.* The rules are arranged in sequence in the policy, and the action of the first rule in the sequence that is matched by a packet is the action executed.
   This is the method usually used in firewalls.
2) *Best Match.* One specific field is chosen; out of the rules matched by the packet, the one that specified the smallest interval for this particular field, wins.
   This is the method used in routing tables - longest prefix matching on the destination IP. (In routing tables, IP address intervals are usually denoted by prefixes, e.g., $100.150.200.0 - 100.150.200.255$, which is really the interval $[1687603200, 1687603455]$, is written $100.150.200.0/24$. Out of the rules matched by the packet, the one with the longest prefix is chosen.)

In practice, precedence in a router is quite complicated:

1) First, find the best match.
2) In case of conflict, choose rules in the order:
   a) Static routes
   b) Dynamic routes, in order (usually EIGRP, OSPF, ISIS, RIP)
   c) Default route
3) If no rule matches, discard the packet.

However, this entire procedure can be effectively reduced to first match, simply by ordering the rules (with prefix length as the primary sort key, and the rule type as the secondary key). Hence, in our work, we assume first match semantics for policies.

The 'winning' rule, the decision of which is implemented for a packet, is said to *resolve* the packet.

### C. OneProb, AllProb, and FieldWidth

This paper focuses on the complexity of policies, as measured by the size of their decision diagrams (as explained in the next subsection). Naturally, the question arises what factors influence the complexity of a policy. The current literature mentions two factors: $n$, the number of rules in the policy (up to several thousand), and $d$, the number of fields in a rule (usually $5 - 10$). However, we suggest that we can get a better sense of the complexity of a policy if we introduce some additional metrics, as follows.

In practice, a rule is almost always very specific - it blocks a particular IP, or allows access to a single port, etc. As a result, most rules have one or two fields of interest set to a single value, and the others set to "All" - i.e. the entire domain of the field. Policies where many fields are set to single values, or to All, show significantly different behavior [14]. The proportion of don't-cares (all-matches) and single values in a policy is captured by the metrics $allprob$ and $oneprob$.

For example, consider the policy

$$\begin{aligned} x_1 \in [1, 100] \wedge x_2 \in [1, 1] &\rightarrow accept \\ x_1 \in [5, 5] \wedge x_2 \in [3, 3] &\rightarrow discard \\ x_1 \in [25, 50] \wedge x_2 \in [1, 50] &\rightarrow discard \\ x_1 \in [10, 10] \wedge x_2 \in [5, 10] &\rightarrow accept \end{aligned}$$

The domain of $x_1$ is [1, 100] and of $x_2$ is [1, 50]. We note that in this ruleset, there are eight fields - two fields each in four rules - and two of these fields, $x_1$ in the first rule and $x_2$ in the third rule, are set to "All". Hence, for this ruleset, $allprob = \frac{2}{8} = 0.25$. Similarly, $oneprob = 0.5$.

Our second observation is that the domains of different fields are of different sizes. For example, IPv4 addresses take 32 bits, protocol takes 8 bits, and version takes 4 bits in a standard header. *Narrow* fields, whose domain is expressed in a small number of bits, affect the policy very differently than other ("wide") fields, as discussed in Section IV. We suggest that only wide fields be counted in $d$, and narrow fields be represented instead by $fieldwidth$, the total number of bits needed to express all the narrow fields in the policy.

### D. Decision Diagrams

A decision diagram is an acyclic, rooted digraph, and a simple DFA representation of a policy. Every non-terminal node, i.e. node with outgoing edges, is marked with the name

of a field, $f_i$. Every edge starts at a non-terminal node (say $f_i$), and is marked with an interval of values, e.g. $[20, 110]$, called the *label* of the edge; the label indicates that, for a packet whose $f_i$ lies in this interval, this edge is to be taken. Every terminal node, i.e. one with no outgoing edges, is marked with a decision. (For example, in case of firewalls, decisions are *accept* or *discard*, which we will henceforth represent as 1 and 0.)

For any packet $p = (p.f_1, ..p.f_d)$ resolved by the policy, there is exactly one path from the root corresponding to the field values of $p$. This path terminates in a terminal node; the decision at this node is the decision of the policy for $p$.

The new concept we add to decision diagrams is *rules in play*. Essentially, a rule $R$ is in play at a node if one or more packets, that match $R$, can reach the node (on their path from the root to a leaf node). A rule is in play along an edge if it is in play at the target node of the edge. At the root, all rules are in play; while traveling from the root to a leaf, the set of rules in play becomes smaller, as rules are eliminated by the choices at nodes; and finally, the first rule still in play at the terminal node "wins", i.e. determines the decision taken.

As decision diagrams are much easier to understand from an example, We present the algorithm in Figure 1, and demonstrate it in Figure 2 with the example policy

$$x \in [10, 110] \wedge y \in [90, 190] \rightarrow 0$$
$$x \in [20, 120] \wedge y \in [80, 180] \rightarrow 1$$

We wish to note two points:

1) As we are concerned with complexity, the question immediately follows - time or space complexity? The answer is, both, as policy verification can have a time complexity equal to the space complexity of the decision diagram. (In the worst case, a policy can specify all possible packets, and make it necessary to check paths from the root to every terminal node.)
2) It is not difficult to annotate a decision diagram with rules in play at each node. In fact, with some minimal changes, the algorithm for generating decision diagrams can be made to output an annotated diagram.

## III. The Size of Decision Diagrams

In this section, we present our study of the (worst-case) size of a policy decision diagram, measured as the number of leaf nodes of the diagram. We begin with the existing upper bound, $O((2n)^d)$.

The intuition behind the bound is that, as the decision diagram is a tree of fixed depth, its size is maximized by making the branching factor of each node as large as possible.

As there are $n$ rules in the policy, there can be a maximum of $2n-1$ outgoing edges for a node. (Each edge must be labeled with at least one interval. The $n$ rules have $2n$ end points - each interval has a start and an end. These $2n$ end points thus divide the domain into a maximum of $2n-1$ intervals. Hence we can have at most $2n-1$ edges from a node.)

Fig. 1: Building annotated decision diagram from policy

**procedure** ADDNEWRULE(Rule $R$, name $R.name$, node $x$)
  **Add** $R.name$ to list of rules in play at $x$
  **if** $x$ is a terminal node **then**
    **Label** $x$ with the decision of $R$.
  **else**
    $i$ is the field of node $x$.
    $R.i$ is the interval for field $i$ in $R$.
    $x.i_1, x.i_2..$ are the values on the edges from node $x$.
    **Build** new paths from $x$, starting with new outgoing edges labeled with the intervals in $R.i - \{x.i_1, x.i_2..\}$. ▷ To each edge, add new nodes and edge labels as per remaining fields in $R$. The terminal node has decision of $R$. These nodes start with only one rule in play, i.e. $R.name$.
    **for** all labels $x.i_k$ **do**
      **if** $x.i_k \cap R.i$ is empty **then**
        **continue**
      **else**
        $y$ is the target of edge labeled $x.i_k$.
        **for** every interval $x.i_{new} \in x.i_k - R.i$ **do**
          **Copy** (subgraph rooted at $y$ ).
          $y'$ is the new copy of $y$ itself.
          **Add** edge $x \rightarrow y'$, labeled $x.i_{new}$.
        **end for**
        **Relabel** edge $x \rightarrow y$ with $x.i_k \cap R.i$.
        $AddNewRule(R, R.name, y)$
      **end if**
    **end for**
  **end if**
**end procedure**

**procedure** BUILDDDIAGRAM(Policy $\{R_1, R_2..R_n\}$)
  **Create** empty node $x$ with no outgoing edges.
  **Label** $x$ with desired field for root.
  **for** $index \leftarrow n...1$, step $-1$ **do**
    $AddNewRule(R_{index}, index, x)$
  **end for**
  **return** Decision Diagram rooted at $x$.
**end procedure**

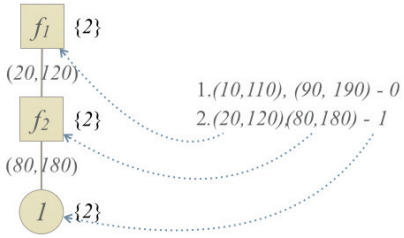
Thus, an upper bound on the size of the decision diagram for a policy of $n$ rules and $d$ fields is $(2n-1)^d$.

Our construction, which is mindful of rules in play, allows us to tighten this bound considerably. The intuition behind this is that, while a node with $n$ rules in play can indeed have a branching factor of $(2n-1)$, not all these branches still have $n$ rules in play. Each child node, with $m$ rules in play, has a maximum outgoing branching factor of $2m-1$ rather than $2n-1$.

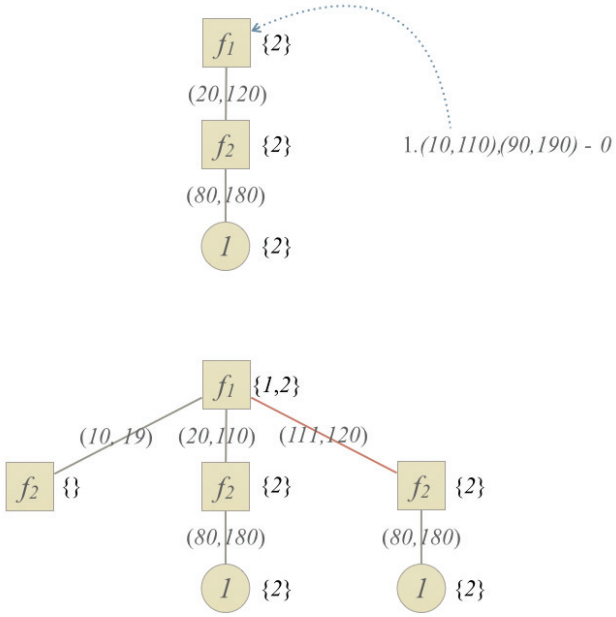
For example, consider the edges emerging from the root in Figure 2. Edge $(10, 19)$ has only one rule in play (the first), edge $(20, 110)$ has both rules in play, and edge $(111, 120)$ has only one rule in play (the second). In general, exactly *one* outgoing edge still has all $n$ rules in play (this is the edge corresponding to the intersection of all the rules). There are also edges with $n-1$ rules in play, $n-2$ rules in play, and so on, down to edges with 1 rule in play.

To provide a worst-case bound, we have to identify what the largest possible decision diagram looks like (for given $n$ and $d$). There are two factors to maximize:

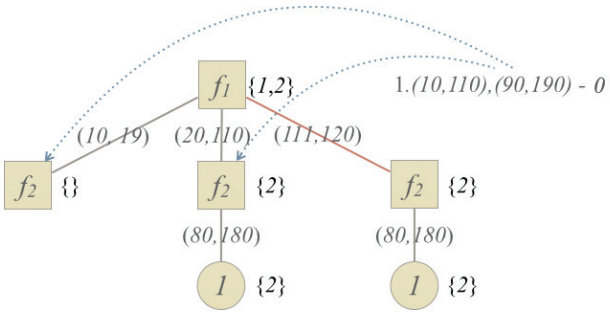
Fig. 2: Example: building a decision diagram, step by step.



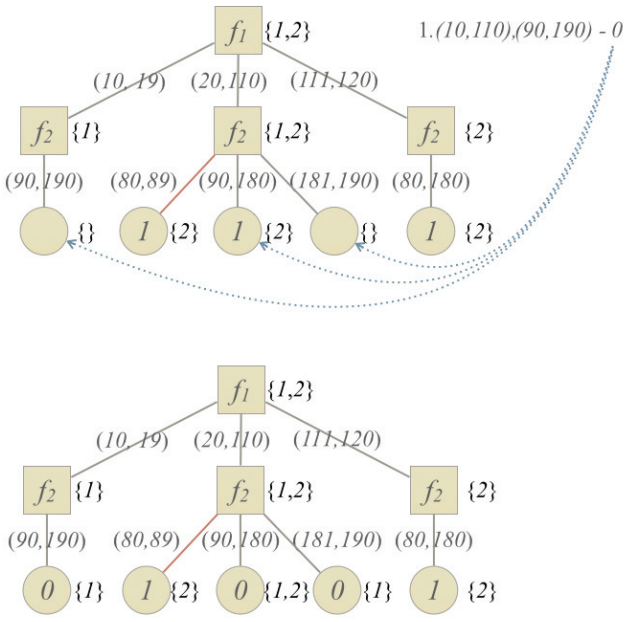

Step 1. First rule from bottom, i.e. rule 2




Step 2. Adding $f_1$ from next rule, rule 1.




Step 3. Adding $f_2$ from rule 1.




Step 4. Adding decision from rule 1.

1) The branching factor at each node.
2) The number of rules in play along each branch. (An edge with more rules in play, leads to more rules in play at the target node, and thus more potential for branching.)

We consider the conditions that hold for the largest decision diagram (for given $n$ and $d$).

**Theorem 1.** *No two rules share the same start values or end values for a field (when the field is a node in the largest decision diagram for some $n$ and $d$) .*

The proof can be given in the form of a game: given a policy where this does not hold, we can always find another - with the same $n$ and $d$ - with a larger decision diagram.

Consider an example ruleset with shared start values.

$$x_1 \in [1, 10] \rightarrow 1$$
$$x_1 \in [20, 30] \rightarrow 0$$
$$x_1 \in [20, 40] \rightarrow 1$$

For any such ruleset, we can edit it to separate the end points, for example by making $start_i$ of rule 3 less than that of rule 2.

$$x_1 \in [1, 10] \rightarrow 1$$
$$x_1 \in [20, 30] \rightarrow 0$$
$$x_1 \in [15, 40] \rightarrow 1$$

This gives us a larger decision tree: we have added a new outgoing edge, and lost nothing. [Note: we are assuming that the domain of the field is large enough to let us separate end points like this. When this assumption does not hold, we have *narrow* fields, which we deal with in Section IV.]

**Corollary 1.1.** *At all non-leaf nodes, the rules-in-play sets of outgoing edges with adjacent labels differ by a single rule (in the largest decision diagram for some $n$ and $d$). [For example, in Figure 2, edges from $f_1$ labeled $(10, 19)$ and $(20, 110)$ have the rules-in-play sets $\{1\}$ and $\{1, 2\}$ respectively. These differ by only one rule: rule 2. ]*

Happily, creating the largest decision tree can be broken up into optimal subproblems. In the largest decision tree, each subtree rooted at a non-leaf node is as large as possible. [Again, we can prove this with a game. Suppose the subtree rooted at a non-leaf node, with $n'$ rules in play and depth $d'$, is *not* the largest tree given $n'$ and $d'$. We simply replace this subtree with the largest, thus increasing the size of the overall tree.] We also know that the size of the largest decision tree cannot decrease as we increase $n$ and $d$.

From the above, we can conclude that in order to create the largest decision tree, we should keep as many rules in play as possible, on the outgoing edges (and thus, at the roots of the child subtrees). [To be precise, when the child subtrees are leaves - i.e., when the depth of the decision tree is 1 - the number of rules in play along the edges does not matter. However, the tree produced by keeping as many rules in play as possible, to the greatest depth possible, is never *smaller* than some other possible tree.]

Let us consider the sequence $S$ of outgoing edges from a node, in increasing order of their labels. (e.g. for the root of Figure 2, and representing edges by their labels, $S$ is ($[10, 19]$, $[20, 110]$, $[111, 120]$) ). Given any rule $R$, $R$ specifies one interval of values (for each field); hence the edges with $R$ in play occur as one continuous sub-sequence in $S$. In other words, once a rule is out of play (by exceeding its $end_i$), it cannot come into play again.

From the above, we know that the largest decision diagram results when we bring rules into play one by one, and take them out of play one by one. To maximize the number of rules in play on the outgoing edges, we should bring rules into play as quickly as possible (i.e. on the first possible edge in sequence $S$), and bring them out of play as slowly as possible (on the last possible edge in $S$).

We are now in a position to state the following theorem.

**Theorem 2.** *In the largest decision diagram rooted at a node with $n$ rules in play, there are $2n - 1$ outgoing edges, and the cardinality of the sets of rules in play along these edges follows the pattern $1, 2, ..., n-1, n, n-1, ...2, 1$.*

From the above two theorems, we see that in the largest possible decision diagram (for given $n$ and $d$), one outgoing edge has $n$ rules in play, two have $n-1$, two have $n-2$, and so on, down to the last two edges which have a single rule in play.

Computing the size of this largest diagram is clearly a simple problem of dynamic programming: given the largest decision diagrams that can be built with $1, 2...n$ rules in play, at a depth $d-1$, we can compute the size of the largest decision diagram with depth $d$, and $n$ rules in play.

We also have the following base cases.

1) For any $d$, when $n = 1$, the number of leaves is 1. (Only a single rule is in play, so there is no branching.)
2) $d = 0$ indicates we are at a leaf node, so the size is 1.

Stating this result formally,

**Theorem 3.** *If we represent the maximum size (measured by number of leaf nodes) of a decision diagram with $n$ rules and $d$ fields as $f(n, d)$,*

$$f(n, d) = 2\sum_{i=1}^{n-1} f(i, d-1) + f(n, d-1)$$
$$\text{for } n > 1, d \geq 1$$
$$f(1, d) = 1$$
$$f(n, 0) = 1$$

This recurrence is very similar to the standard recurrence for trinomial coefficients (*Delannoy numbers*):

$$D(n+1, d+1) = 2\sum_{k=0}^{n} D(k, d) + D(n+1, d)$$
$$\text{for } n \geq 0, d \geq 0$$
$$D(0, d) = 1$$
$$D(n, 0) = 1$$

In fact, the two recurrences are identical if we change the variables slightly:

$$f(n,d) = D(n-1,d)$$

which gives us a new bound on the size of decision diagrams.

Our bound is tight: we know exactly the policy with this decision diagram (and which, therefore, matches the size of this upper bound).

$$x_1 \in [1, 2n-1] \wedge ... \wedge x_d \in [1, 2n-1] \rightarrow 1$$
$$x_1 \in [2, 2n-2] \wedge ... \wedge x_d \in [2, 2n-2] \rightarrow 0$$
$$x_1 \in [3, 2n-3] \wedge ... \wedge x_d \in [3, 2n-3] \rightarrow 1$$
$$...$$
$$x_1 \in [n, n] \wedge ...x_d \in [n, n] \rightarrow 0$$

## IV. Real Decision Diagrams: New Metrics

In the previous section, we presented our new bound for the size of decision diagrams. This bound, the Delannoy number $D(n,d)$, is much smaller than the old bound of $(2n-1)^d$; for example, for $n = 1000$ and $d = 10$,

$$f(n,d) = 2.808 \times 10^{26} << (2n-1)^d = 1.019 \times 10^{33}$$

However, this is still too large, and the bound is tight so there is no room to reduce it further. There must be more reasons why decision diagrams do not grow intractably large in practice. We suggest that the answer can be found in our concepts of oneprob, allprob, and narrow fields.

### A. OneProb and Singletons

In constructing a decision diagram, we have two main operations that increase its size.

1) *Adding a new path.* This happens when the new rule specifies new values for a field, i.e. values for which no outgoing edges exist.
2) *Splitting an edge.* When the interval specified by the rule only partly overlaps with the label of an edge, we 'split' the edge. The entire subtree below the edge is copied. For example, in Figure 2, the new rule has $f_1 \in [10, 110]$ and the old edge has $f_1 \in [20, 120]$. They partially overlap. We get new edges labeled $f_1 \in [10, 19]$, $f_1 \in [20, 110]$, and $f_1 \in [111, 120]$.

Now we consider the impact of *singletons*. A rule is called a singleton for field $f_x$ if it matches only packets with one single value of $f_x$, i.e. its interval for $f_x$ is a single value like $[29, 29]$.

It is important to note that a singleton rule at a node is no different than any other rule, at the lower levels. We only know that it is a singleton for that one field; this says nothing about the other fields - it is not at all necessary that the rule is also a singleton for the fields at lower levels! So as a rule in play, it is just as powerful as any other rule.

The power of a singleton for field $f_x$ is seen at the $f_x$-labeled nodes. As it covers a single value for $f_x$, it cannot partially overlap with any other rule. So an edge from an $f_x$ node labeled with a singleton (for $f_x$) cannot be split.

We immediately observe that if all the rules are singletons, the worst-case branching factor at the node drops from $2n-1$ to $n$ (when all the singletons have different values for $f_x$, and thus produce different branches). However, this attractive idea is of limited power: even one non-singleton rule can return the branching factor to $2n-1$. For example, consider rules $R_1, R_2, R_3$ with respective intervals $[1, 10]$, $[3, 3]$ and $[7, 7]$. We have $2 \times 3 - 1 = 5$ edges:

1) $[1, 2]$ (with $R_1$ in play)
2) $[3, 3]$ ($R_1, R_2$ in play)
3) $[4, 6]$ ($R_1$ in play)
4) $[7, 7]$ ($R_1, R_3$)
5) $[8, 10]$ ($R_1$)

To see how having singleton rules (rather than non-singleton rules) makes the decision diagram smaller, we consider the number of rules in play along an edge. We will use the new function $g(n,d)$ to denote the size of decision diagram in presence of singletons.

Consider an $f_x$-labeled node, with $s$ singletons (and $t$ non-singleton rules). The singletons either overlap completely, or they do not overlap at all.

1) If they do overlap, this reduces the branching factor at the node: there are fewer outgoing edges.
2) If they do not overlap, the number of rules in play along the outgoing edges is less. Hence the size of the subtrees below the node is reduced.

For the largest decision diagram, we need to trade off the "immediate" branching factor at the node and the "potential" for more branching at lower levels. The solution is only trivial for the last field, i.e. $d = 1$: all singletons for the last field have distinct values. This is because the only lower level is the leaf nodes - there can be no branching at a lower level.

Singletons increase the size of a decision diagram most effectively if they split the outgoing edge with the largest possible number of rules in play. Consider the process of adding a singleton for field $x$ to any decision diagram. If the singleton does not split any edge, it introduces only one new edge, where it is the only rule in play. If it splits an edge with $r$ rules in play, one edge with $r$ rules is replaced by two edges with $r$ rules (before and after the singleton), plus one edge with $r+1$ rules (the overlap with the singleton). The maximum is when $r = t$.

Our task is simplified by the fact that the largest decision diagram is produced by adding singletons to the largest diagram without singletons. The reason is the monotonicity of the size function. With or without singletons, a diagram with more edges and more rules in play along an edge is larger. To make this clear, we present an example.

- Consider two diagrams, where the number of rules-in-play are $(1, 2, 1, 2, 1)$ and $(1, 2, 3, 2, 1)$. We add two singletons to each of these.
- For any diagram $A$, the size of $A$ grows fastest by adding singletons that split the edge with the most rules in play.

- The first decision diagram only has edges with up to 2 rules in play. After adding the singletons, the maximum size for the first decision diagram is

$$\begin{aligned} &g(1,d)+\\ &max\big(g(4,d)+2g(2,d),2g(3,d)+3g(2,d)\big)+\\ &g(1,d)+g(2,d)+g(1,d) \end{aligned}$$

- For the second decision diagram, the maximum size is

$$\begin{aligned} &g(1,d)+g(2,d)+\\ &max\big(g(5,d)+2g(3,d),2g(4,d)+3g(3,d)\big)+\\ &g(2,d)+g(1,d) \end{aligned}$$

  which is clearly larger.

Generalizing from the example, we see that the largest decision diagram without singletons not only starts out as the largest, but also produces the largest increases, as it has an edge with $t$ rules in play; no other diagram can do better.

There remains the question of whether the singletons themselves should overlap, or not. This is not straightforward. Let us assume that for each field the number of singletons is $s$, the number of non singletons is $t$ (i.e. $n = s + t$), and the depth of the tree (below the node we are considering) is, as usual, $d$. Then to find the size of the largest decision diagram, we try all the partitions of $s$ singletons; for example, if $s = 4$, we need to try $1+1+1+1, 1+2+1, 1+3, 2+2$, and $4$. Given $g(n, d)$ is the formula for maximum size, $n = s + t$, and that we partition the singletons into $a$ groups $\{s_1, s_2, ...s_a\}$,

$$\begin{aligned} g(n,d) =& 2\sum_{i=1}^{t-1} g(i,d-1)\\ &+ max\Big[(a+1)g(t,d-1)\\ &+\sum_{j=1}^{a} g(t+s_j,d-1)\Big]\\ n =& s+t \end{aligned}$$

The "max" of the formula is taken over all partitions. In other words, we compute the size for all possible partitions, according to the formula; the maximum is taken.

The probability that a rule is a singleton is given by $oneprob$, so our intuition is that a high value of $oneprob$ leads to a smaller decision diagram. [Singletons are also much less likely to overlap than other rules are. As we are performing worst-case analysis, we do not make use of this factor, but it most likely also plays a role in the tractable size of practical decision diagrams.] We explore the impact of singletons in the Results Section.

### B. AllProb and All-Matches

The behavior of *all-matches*, that is, rules that match all values for a given field, is simple. At a given node, every all-match rule covers the entire domain for the field. So the all-matches all behave like one single large rule, which adds $u$ (the number of all-match rules) to the number of rules in play on each edge. We can update the formula for largest decision diagram as follows.

$$\begin{aligned} g(n,d) =& 2\sum_{i=0}^{t-1} g(i+u,d-1)\\ &+ max\Big[(a+1)g(t+u,d-1)\\ &+\sum_{j=1}^{a} g(t+s_j+u,d-1)\Big]\\ n =& s+t+u \end{aligned}$$

where as in the previous section, the singletons are partitioned into $a$ groups named $s_1...s_a$, and the maximum size is taken over all such partitions.

Intuitively, the size of the decision diagram is reduced by increasing the proportion of all-match rules, as most of the rules have total (rather than partial) overlap. In the extreme case where all the rules are all-matches, the first rule resolves all packets, and the decision diagram becomes a linked list running straight from the root to the decision of the first rule. The impact of allprob, the proportion of rules that are all-matches, is experimentally shown in the Results Section.

### C. Narrow Fields

In our construction of decision diagrams, we state that the outgoing edges from a node can carve the domain into $2n-1$ pieces. This statement made the unspoken assumption that the domain is large enough to divide into so many distinct pieces. However, this assumption is not true for some domains. These are the ones we call *narrow* fields.

A field $f_w$ represented with $w$ bits has a domain of $2^w$ values, so the number of outgoing edges from a $f_w$ node is upper bounded by $2^w$. [Every possible value of the field labels a distinct edge.] A narrow field is one where this is the tighter upper bound, i.e. $2^w < 2n - 1$.

To see the impact of narrow fields on the decision diagram, we place all the narrow fields together, as the fields closest to the leaves (i.e. with $d = 1, 2...d_{narrow}$. Suppose the fieldwidth is $W$ bits, i.e. the narrow fields can all be represented using $W$ bits. Our computation for the size of the decision diagram does not change for the non-narrow fields; however, the "leaf" nodes for this tree are now the roots of decision trees for the narrow fields, which expand to $2^W$ bits each.

We compute the size of the upper tree (without narrow fields) as per the previous formula, and multiply the result by $2^W$ to get an upper bound on the size. This idea is similar

to the automata size bound of Erradi [15].

$$\begin{aligned} g(n,d) =& 2\sum_{i=0}^{t-1} f(i+u, d-1) \\ &+ max\Big[(a+1)g(t+u, d-1) \\ &+ \sum_{j=1}^{a} g(t+s_j+u, d-1)\Big] \\ n =& s+t+u \end{aligned}$$

for $d > d_{narrow}$, the number of narrow fields, and

$$g(n,d) = g(n, d-1) * 2^{w_d}$$

otherwise. ($w_d$ = bit width of field at height $d$.)

However, this is no longer a *strict* upper bound. Consider our decision diagram, with wide fields closer to the root and narrow fields closer to the leaves. There exist edges from the upper nodes to the lower nodes with a very small number of rules in play - 1, 2 etc. For these edges, the lower nodes are no longer "narrow" fields, as the bound of $2n - 1$ is again smaller than $2^w$.

Though we do not have a strict bound in the presence of narrow fields, their existence also serves to reduce the size of the decision diagram considerably, as can be seen in the Results section.

## V. Minimum Equivalent Policies

In this section, we discuss a completely new application of our annotated decision diagrams. Keeping track of rules in play allows us to take any policy and find a $minimum$ equivalent policy, i.e., the smallest policy with the same decision for every packet as the original policy. (To be strictly accurate, we find the minimum equivalent policy that can be expressed with rules present in the original policy.) This is a much stronger statement than the minimality guarantees of algorithms such as Probe [16]; these simply find a $minimal$ policy, i.e. one with no redundant rules. For example,

$$\begin{aligned} x \in [1,100] \wedge y \in [1,100] &\rightarrow 0 \\ x \in [1,50] \wedge y \in [1,100] &\rightarrow 0 \\ x \in [51,100] \wedge y \in [1,100] &\rightarrow 0 \end{aligned}$$

If we delete the first rule, we get a policy that is minimal, but not minimum (as we could have kept the first rule and deleted the other two to get a smaller, equivalent policy).

The intuition behind our algorithm, LeafTrim, is simple. We delete rules under the constraint that we cannot change the decision of the policy for any packets. The decision diagram may change; some branches may merge, so different leaf nodes become a single node. (As we never add rules, there will be no new paths or new leaves to consider.) But the decision cannot change for any packet - so we cannot make a change that changes the decision at any leaf node.

In brief, to ensure that the decision at a leaf node is preserved, we must keep at least one rule $R$ with this decision, and also make sure that any conflicting rules that precede $R$ are removed. Using the Boolean variable $p_i$ to mean that rule $i$ is present, we can express this as a logical constraint.

For example, consider a leaf node with the $accept$ rules $3, 13$ and the $discard$ rule 7 in play. As long as we do not delete rule 3, deleting other rules does not affect the decision at this leaf. What if we delete rule 3? We must delete rule 7 (so it does not become the first rule to match these packets), and keep rule 13. In other words, the decision of the leaf remains unchanged iff the following formula holds.

$$p_3 \vee (\neg p_7 \wedge (p_{13}))$$

For a more complex case, with more rules in play, we can apply the same logic recursively. For example, suppose we have $accept$ rules 3, 13 and 23, and $discard$ rules 7, 10 and 17.

- If we delete rule 3, we must delete 7 and 10.
- If we also delete rule 13, we must delete 17.
- If we have deleted both 3 and 13, we cannot delete 23. We cannot preserve the policy decision for packets resolved at this leaf, if we delete *all* the rules with this decision. (Here, the decision was $accept$, as the first rule - rule 3 - was an $accept$ rule.)

The constraint formula for the leaf is

$$p_3 \vee (\neg p_7 \wedge (\neg p_{10} \wedge (p_{13} \vee (\neg p_{17} \wedge (p_{23})))))$$

We now present the algorithm to find the constraint formula for a leaf node.

1) At a leaf node, we start with the first rule in play. All other rules with the same decision are *complying* rules, and the others are *conflicting* rules.
2) We traverse the list of rules in play, in order, building an expression (string), as follows :
   - If the rule $R_i$ is complying, we add "$p_i \vee$ (".
   - If it is conflicting, we add "$\neg p_i \wedge$ (".
   - At the end, we close all parentheses.

The final step is to notice that, to preserve the semantics of the entire policy, it is both necessary and sufficient to return the same decision for all packets - no matter which leaf node they are resolved at.By taking the AND of the expressions for all the leaf nodes, we get the constraint expression which must be satisfied to keep the decision of the policy unchanged for all packets.

To find the minimum policy, we take the constraint expression for the original policy (as described above), and search for the solution with the smallest number of $p_i$ set to 1. In other words, we have reduced our original problem (finding the smallest subset of rules that will preserve the semantics of the policy) to the Min-One SAT problem. This can now be solved with a standard fast SAT solver; we use OptSAT [17].

The complete algorithm for computing a minimum equivalent policy is formally presented in Figure 3. In the next section, we go on to discuss our experimental results, including the performance of this algorithm.

Fig. 3: LeafTrim Algorithm for Minimum Equivalent Policy.

**procedure** LEAFCONSTRAINTS(decision tree D)
  **Create** empty string $allConstraints$
  **for** leaf in decision tree D **do**
    **Create** string $leafConstraints$ = "("
    Sort rules in play at leaf by priority to get LeafRules
    $DecisionOfLeaf$ = decision of highest-priority rule in leaf (i.e. first rule in LeafRules)
    **for** $rule_i$ in LeafRules **do** ▷ $i$ is the index of $rule_i$ in the original policy
      **if** decision of rule == $DecisionOfLeaf$ **then**
        $leafConstraints = leafConstraints +$ "$p_i \vee$ ("
      **else**
        $leafConstraints = leafConstraints +$ "$\neg p_i \wedge$ ("
      **end if**
    **end for**
    Close all open brackets in $leafConstraints$
    $allConstraints = allConstraints +$ " $\wedge$ " $+ leafConstraints$
  **end for**
  **return** $allConstraints$.
**end procedure**

**procedure** LEAFTRIM(policy $P$)
  $DDiagram = BuildDDiagram(P)$
  $Constraints = LeafConstraints(DDiagram)$
  $RulestoKeep = MinOneSAT(Constraints)$
  policy $P'$ = rules of $P$ named in $RulestoKeep$
  **return** $P'$
**end procedure**

## VI. EXPERIMENTAL RESULTS

In this section, we begin by showing, experimentally, how our new metrics affect decision diagram size for practical policies ($n = 100, 200...1000$ rules and $d = 5$ fields). We start by showing the original bound $f(n, d)$, then $g(n, d)$, our new bound; next, we introduce our mitigating factors, introducing first $allprob$, then narrow fields, and finally $oneprob$. The results are shown in Figure 4.

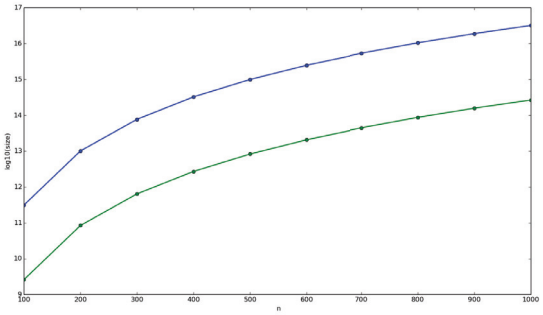
Our second set of experimental results are concerned with the LeafTrim algorithm for redundancy removal. We are concerned about two issues:

1) *Effectiveness.* How powerful is the new algorithm - is there a noticeable difference in the size of a minimum policy, as computed by LeafTrim, and a minimal policy computed by a state-of-the-art algorithm (here, we have used Probe [16])?
2) *Cost.* Is the algorithm feasible to run for large policies?

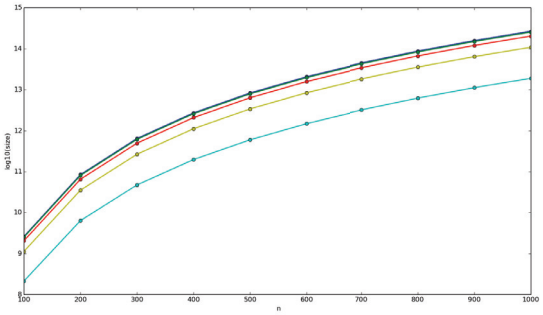
We are happy to report that the algorithm, though very much slower than Probe, is far more effective in terms of the size reduction. However, we are only able to run it for small values

Fig. 4: Performance of size bounds for Decision Diagrams

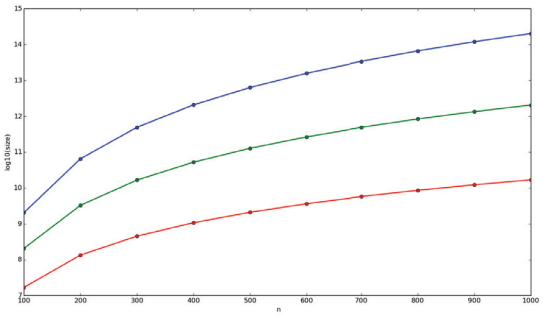
(a) First bound: smaller than $(2n - 1)^d$



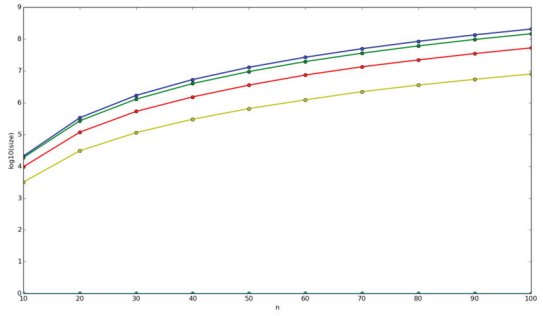
(b) Effect of $allprob$. (Top to bottom, $0, 20, 40, 60, 80$.)



(c) Effect of narrow fields. $allprob = 40$. (Lines, top to bottom, 5 wide fields; 4 wide + 1 narrow; and 3 wide + 2 narrow. Narrow fields 2 bits wide.)



(d) Effect of $oneprob$. (Top to bottom, $0, 20, 40, 60$.) $allprob = 40$, 4 wide + 1 narrow field.



of $n$ (50 and 100 in our tests); for $n = 200$ the algorithm took 21.6 GB of virtual memory and crashed. LeafTrim, at least in its naive form, is not yet scalable to large policies.

In our tests, we took the average of 400 observations each, using generated firewalls with $d = 5$, one narrow field (2 bits), and a width index of $(40, 40)$ for the tests, which is in line with the practical firewalls we have observed.

A brief digest of our results is as follows.

| Length | Time(sec.) | Reduction(%age) |
|---|---|---|
| n = 50 | 80 | 48 |
| n = 100 | 3185 | 61 |

We add that the SAT solver OptSAT was extremely fast; its running time was less than $0.5\%$ of the total. The reported time was expended almost entirely in constructing the FDD and building the constraint formula for solution.

For comparison, here are the corresponding figures for a fast state-of-the-art algorithm, Probe [16], on the same policies. Clearly, redundancy removal with Probe is much faster, but far less effective.

| Length | Time(sec.) | Reduction(%age) |
|---|---|---|
| n = 50 | 0.081 | 12.5 |
| n = 100 | 0.292 | 16 |

As redundancy removal is an offline process (we only need to remove redundancy once, and not repeatedly while using the policy), a stronger algorithm like LeafTrim is clearly a better choice. However, as it stands, LeafTrim is still not scalable to a large policy. Our future work will focus on the question of whether it can be optimized and scaled up.

## VII. Concluding Remarks

In this paper, we introduce the concept of "rules in play", and make two contributions to the theory of network policies. First, we give a tight upper bound on the size of policy decision diagrams, and propose some new metrics (oneprob, allprob and fieldwidth) that may explain why the size does not grow explosively for practical policies. We contend that decision diagram based algorithms are tractable in practice because their running time is not only bounded by $O(n^d)$, as was previously thought, but also constrained by these new metrics; practical policies have tractable values for these metrics.

Our second contribution is LeafTrim, the first optimization algorithm that can produce a truly minimum-length policy. The algorithm is only tractable for small policies, but in our tests, dramatically improves on earlier algorithms that can only produce a minimal policy (one with no redundant rules).

Our work on decision diagrams suggests several problems for further study. For example, how do our metrics influence other algorithms and data structures for network policies, such as decision-diagram compression by Bit Weaving [18]? Can our new algorithm for policy optimization be scaled up to large policies? We intend to explore these questions in our future work.

## References


[1] P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds," in *Proceedings of IEEE INFOCOM*, 1998.

[2] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed ip routing lookups," in *Proceedings of ACM SIGCOMM*, 1997, p. 2536.

[3] S. Suri, G. Varghese, and P. Warkhede, "Multiway range trees: Scalable ip lookup with fast updates," in *GLOBECOM*, 2001.

[4] S. Sahni and K. Kim, "O(log n) dynamic packet routing," in *IEEE Symposium on Computers and Communications*, 2002.

[5] A. X. Liu and M. G. Gouda, "Firewall policy queries," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 6, pp. 766–777, 2009.

[6] E. Al-shaer, W. Marrero, A. El-atawy, and K. Elbadawi, "Network configuration in a box: Towards end-to-end verification of network reachability and security," 2009.

[7] H. H. Hamed, E. S. Al-Shaer, and W. Marrero, "Modeling and verification of ipsec and vpn security policies," in *ICNP*, 2005, pp. 259–278.

[8] D. Eppstein and S. Muthukrishnan, "Internet packet filter management and rectangle geometry," in *SODA*, 2001, pp. 827–835.

[9] M. Frantzen, F. Kerschbaum, E. E. Schultz, and S. Fahmy, "A framework for understanding vulnerabilities in firewalls using a dataflow model of firewall internals," *Computers & Security*, vol. 20, no. 3, pp. 263–270, 2001.

[10] D. Hoffman and K. Yoo, "Blowtorch: a framework for firewall test automation," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005, pp. 96–103.

[11] H. B. Acharya and M. G. Gouda, "Firewall verification and redundancy checking are equivalent," in *INFOCOM*, 2011, pp. 2123–2128.

[12] D. Shah and P. Gupta, "Fast updating algorithms for tcams," *IEEE MICRO*, vol. 21, no. 1, p. 3647, 2001.

[13] A. Basu and G. Narlika, "Fast incremental updates for pipelined forwarding engines," in *Proceedings of IEEE INFOCOM*, 2003.

[14] H. B. Acharya, "On rule width and the unreasonable effectiveness of policy verification," in *IEEE 39th Conference on Local Computer Networks, LCN 2014, Edmonton, AB, Canada, 8-11 September, 2014*, 2014, pp. 314–321.

[15] A. Khoumsi, W. Krombi, and M. Erradi, "A formal approach to verify completeness and detect anomalies in firewall security policies," in *Foundations and Practice of Security*, 2014, pp. 221–236.

[16] H. B. Acharya and M. G. Gouda, "Projection and division: Linear-space verification of firewalls," *Distributed Computing Systems, International Conference on*, pp. 736–743, 2010.

[17] E. Giunchiglia and M. Maratea, "Optsat: A tool for solving sat related optimization problems." 2006.

[18] C. R. Meiners, A. X. Liu, and E. Torng, "Bit weaving: A non-prefix approach to compressing packet classifiers in tcams," *IEEE/ACM Trans. Netw.*, vol. 20, no. 2, pp. 488–500, 2012.