Minflate: Combining Rule Set Minimization with Jump-based Expansion for Fast Packet Classification

— DRAFT —

Sven Hager[†] Patrik John[†] Andreas Fiessler[‡] Björn Scheuermann[†]

[†]Humboldt University of Berlin, Germany
{hagersve,john,scheuermann}@informatik.hu-berlin.de

[‡]genua mbH, Germany
andreas_fiessler@genua.de

ABSTRACT

Network packet classification is a key functionality for packet filters and firewalls, and its performance is crucial for such systems to maintain a high packet throughput under heavy load situations. However, many existing packet filters employ slow classification algorithms which cannot provide the required lookup performance due to slow rule set traversal. In this work, we address this problem by providing a novel rule set transformation strategy called *Minflate* which combines the advantages of existing orthogonal transformation schemes by first minimizing a source rule set and then encoding decision trees into the minimized rule set. Our results show that the Minflate-generated rule sets are both small and can in many cases be traversed faster than rule sets transformed by existing techniques in isolation.

1. INTRODUCTION

Network packet classification is a key building block for packet filters and firewalls, such as the widely deployed Linux' `iptables` and FreeBSD's `ipfw`. The primary task of these services is to discriminate incoming network packets based on certain header fields with respect to a predefined *rule set* in order to establish a security policy, enable traffic rate limiting, or implement QoS routing. However, if the size of the implemented rule set grows, these systems can easily be brought to their knees in terms of packet throughput, as the classification performance does not meet the line speed requirement. The reason for this behavior is twofold: first, many widely used systems, such as `iptables` and `ipfw`, still employ a basic linear search algorithm to traverse the rule set for every single packet—although there are significantly faster algorithms at hand [4]. Second, these systems often do not properly optimize the installed rule set at load time to reduce the number of processed rules during the packet classification process.

In this work, we propose a novel rule set transformation approach called *Minflate* that significantly increases the classification performance of existing packet filtering systems *without* the need to adapt their current implementation. By composing existing orthogonal rule set transformation techniques, which either minimize [2] or inflate a specified source rule set in a controlled way [1], we generate an optimized rule set which is semantically equivalent to the source rules, but can be traversed significantly faster than rule sets generated by existing transformation techniques in isolation. Our evaluation indicates that the Minflate approach requires sub-second preprocessing time for rule sets of up to 5,000 rules. Furthermore, our throughput experiments with `iptables` and `ipfw` show that the Minflate-generated rule sets can be traversed over 13 times faster than an original source rule set, and over 1.2 times faster than a rule set generated by existing optimization techniques.

ANCS '16 March 17-18, 2016, Santa Clara, CA, USA

ACM ISBN 978-1-4503-4183-7/16/03.

DOI: http://dx.doi.org/10.1145/2881025.2889485

2. PROBLEM STATEMENT

We assume that a *rule set* $\mathscr{R}$ with N *rules* R_i is an ordered collection $\mathscr{R} = < R_1, \dots, R_N >$. Each rule R_i consists of an *action* A_i and one or more checks C_j which are executed on specific header fields of an incoming packet P, e.g., $\texttt{dst\_port}(P) = 80$ or $\texttt{src\_addr}(P) \in 101.20.0.0/16$. The goal of the packet classification problem is to find the smallest index i^* for a packet P where all checks in rule R_{i^*} match the corresponding header fields in P. Then, the associated action A_{i^*} (e. g., DROP or ACCEPT) is executed.

3. THE MINFLATE APPROACH

The proposed Minflate technique is a source-to-source rule set optimization approach that transforms an input rule set $\mathscr{R}$ into a semantically equivalent output rule set $\mathscr{R}'$, which can be traversed faster by the underlying packet filter. To this end, Minflate combines the existing rule set optimization strategies *Firewall Compressor* [2] and *HiTables* [1], which implement orthogonal optimization strategies in order to achieve the goal that fewer rules have to be processed by the packet filter at run time. The Minflate transformation flow is sketched in Figure 1.

Firewall Compressor, on the one hand, aims to minimize the overall number of rules in the source rule set by decomposing and merging selected rules. The resulting output rule set typically has fewer total rules and thus can often be traversed faster. HiTables, on the other hand, exploits the fact that many existing packet filters, such as `iptables` and `ipfw`, provide the ability to conditionally redirect the packet classification control flow by so called *jump rules* [1]. These jump rules are used by HiTables in order to integrate decision tree search structures [3] into the given source rule set. During the rule set traversal, the jump rules dispatch on

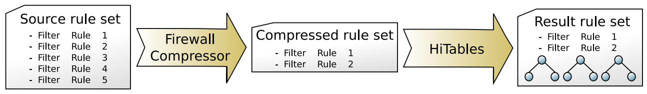
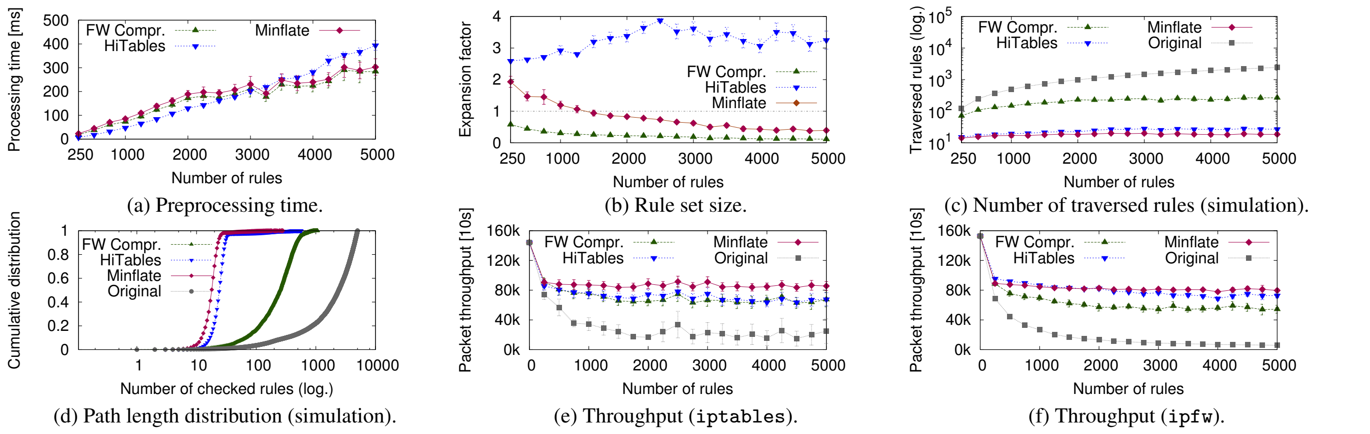


Figure 1: The Minflate approach.



(a) Preprocessing time. (b) Rule set size. (c) Number of traversed rules (simulation).

(d) Path length distribution (simulation). (e) Throughput (`iptables`). (f) Throughput (`ipfw`).

Figure 2: Evaluation results.

the packet's header fields to quickly reach the highest prioritized matching rule. Although this process increases the rule set size, it can be traversed faster because many rules that would have been tested in a linear search are skipped.

The key idea behind Minflate is a functional composition of Firewall Compressor and HiTables in order to combine the advantages of both approaches: a small size of the output rule set as well as decision trees for fast traversal. To this end, Minflate computes the output rule set $\mathscr{R}'$ by chaining the Firewall Compressor and HiTables techniques, i. e., $\mathscr{R}' = \text{hitables}(\text{fw\_compressor}(\mathscr{R}))$. Thus, the input rule set $\mathscr{R}$ is first transformed to a compressed rule set $\mathscr{R}_C$, and subsequently translated to the output rule set $\mathscr{R}'$ which contains decision trees for fast traversal.

4. EVALUATION

We evaluated the efficiency of Minflate in terms of preprocessing time, the number of rules in the output rule set, and matching performance both in a simulated environment as well as with generated network traffic on the `iptables` and `ipfw` packet filters. To this end, we used the ClassBench benchmark tool [5] in order to generate rule sets with sizes ranging from 250 up to 5,000 rules, in steps of 250. For each rule set size, we generated 20 different rule sets. Also, for each rule set we generated a corresponding trace of 100,000 packet headers uniformly distributed over the rule set. Our measurement results are shown in Figure 2. All values are averaged over the 20 test runs for each rule set size, and the error bars represent 95% confidence intervals.

Figure 2a shows the preprocessing times for C++ implementations of the Minflate, Firewall Compressor, and HiTables techniques on an Intel Xeon E7 2.5 GHz machine running Linux. It can be seen that Minflate has the highest preprocessing times for rule set sizes up to 3,000. For larger rule sets, Minflate executes faster than HiTables, as the rule set compression in the first step enables a faster decision tree generation in the second step.

Next, Figure 2b depicts the size of the transformed rule sets in terms of a factor to the size of the source rule set. While Firewall Compressor reduces the overall size, the HiTables-generated rule set is enlarged, which is due to the decision trees encoded in the generated rule set. We also see that Minflate enlarges the compressed rule set, but still generates rule sets smaller than the source rule set for more than 1,500 rules.

Figures 2c shows the mean number of traversed rules in the rule sets. Furthermore, Figure 2d exhibits the distribution of the number of traversed rules, for rule sets of 5,000 rules. These results were obtained by matching the generated headers against the corresponding rule sets in a simulated environment. The plots reveal that the Minflate-generated rule sets classify the given headers by traversing the least amount of rules.

Finally, Figures 2e and 2f show our throughput results using `iptables` and `ipfw`. Here, we used `tcpreplay` to loop over the generated traces for 10 seconds and send minimal-sized packets to Raspberry Pi 1 machines, which ran Linux with `iptables` and FreeBSD with `ipfw`, respectively. The number of processed packets after 10 seconds were counted with `netstat`. In both figures, we observe a sharp throughput drop between 0 and 250 rules, which demonstrates the large overhead induced by the packet classification system. However, the figures also show that, while the throughput for the original rule sets continues to drop with increasing rule set sizes, the optimization techniques are able to significantly improve the classification performance. In case of `iptables`, Minflate performs always best for each evaluated rule set size, outperforming HiTables by a factor of over 1.2. For `ipfw`, HiTables gives better results for small rule set sizes up to 2,000. However, for larger rule sets, Minflate outperforms HiTables by a factor of up to 1.1. The reason for the slightly worse results for small rule sets is an inefficient representation of IP ranges in `ipfw`.

5. CONCLUSION

We presented Minflate, a novel rule set optimization technique which combines the benefits of existing transformation schemes through functional composition. Minflate first minimizes a source rule set with Firewall Compressor and subsequently augments it with decision trees using HiTables to generate output rule sets which can be quickly searched by the underlying packet filter. Our simulation- and traffic-based experiments confirm that the Minflate-generated rule sets can in most cases be traversed faster than rule sets generated by the above-mentioned techniques in isolation.

6. REFERENCES

[1] S. Hager, S. Selent, and B. Scheuermann. Trees in the list: Accelerating list-based packet classification through controlled rule set expansion. In *CoNEXT '14*, pages 101–107, Dec. 2014.

[2] A. Liu, E. Torng, and C. Meiners. Firewall compressor: An algorithm for minimizing firewall policies. In *INFOCOM '08*, pages 691–699, Apr. 2008.

[3] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li. Packet classification algorithms: From theory to practice. In *INFOCOM '09*, pages 648–656, Apr. 2009.

[4] D. Taylor. Survey and taxonomy of packet classification techniques. *ACM Comput. Surv.*, 37(3):238–275, Sept. 2005.

[5] D. Taylor and J. Turner. ClassBench: a packet classification benchmark. *IEEE/ACM Transactions on Networking*, 15(3):499–511, June 2007.