THE GEOMETRIC EFFICIENT MATCHING ALGORITHM FOR FIREWALLS
EXTENDED ABSTRACT

Dmitry Rovniagin, Avishai Wool

School of Electrical Engineering,
Tel Aviv University,
Ramat Aviv 69978, Israel.
jinn@post.tau.ac.il, yash@acm.com

ABSTRACT

Firewall packet matching can be viewed as a point location problem: Each packet (point) has 5 fields (dimensions) which need to be checked against every firewall rule in order to find the first matching rule. In this paper we consider a packet matching algorithm, which we call the Geometric Efficient Matching (GEM) algorithm. The GEM algorithm enjoys a logarithmic matching time performance, easily beating the linear time required by the naive matching algorithm. However, the algorithm's theoretical worst-case space complexity is $O(n^4)$ for a rule-base with n rules.

Based on statistics from real firewall rule-bases, we created a model that generates random, but non-uniform, rule-bases. We evaluated GEM via extensive simulation using this rule-base generator. Subsequently, we integrated GEM into the code of the Linux `iptables` open-source firewall. Our `GEM-iptables` implementation supports a throughput which is at least 5-10 times higher than that of the unoptimized `iptables`. Our implementation was able to match over 30,000 packets-per-second even with 10 thousand rules.

1. INTRODUCTION

The firewall is one of the main technologies allowing high-level access control to organization networks. Firewall packet matching involves matching on many fields from the TCP and IP packet header. With available bandwidth increasing rapidly, very efficient matching algorithms need to be deployed in modern firewalls, to ensure that the firewall does not become a bottleneck.

Most modern firewalls are stateful. This means that after the first packet in a network flow is allowed to cross the firewall, all subsequent packets belonging to that flow, and especially the return traffic, is also allowed through the firewall. This statefulness is commonly implemented by two separate search mechanisms: (i) a slow algorithm that implements the "first match" semantics and compares a packet to all the rules, and (ii) a fast state lookup mechanism that checks whether a packet belongs to an existing open flow. In many firewalls, the slow algorithm is a naive linear search of the rule-base, while the state lookup mechanism uses a hash-table or a search-tree.[1] Note that this two-part design works best on long TCP connections, for which the fast state lookup mechanism handles most of the packets. However, connectionless UDP[2] and ICMP traffic, and short TCP flows (like those produced by HTTP v1.0), only activate the "slow" algorithm, making it a significant bottleneck. Our main result is that the "slow" algorithm does *not* need to be slow. We show that the GEM algorithm has a matching speed that is comparable to that of the state lookups. Our Linux `GEM-iptables` implementation sustained a matching rate of over 30,000 pps[3], with 10,000 rules, without losing packets.

1.1. Related Work

There is an extensive literature dealing with router packet matching. Existing algorithms implement the "longest prefix match" semantics, using several different approaches.

The IPL algorithm of [3], which is based on results introduced in [4], divides the search space into elementary intervals by different prefixes for each dimension, and finds the best (longest) match for each such interval.

The Tuple Space Search algorithm is described in [5] and it's extension in [6]. In this algorithm, all the prefixes are divided into tuples by field prefix length, and then searched linearly. To reduce the time complexity, the authors use pre-computations, markers and heuristic decisions based on statistics of tuples sizes. Other packet matching algorithms include Line Search on multi-dimensional tuple space [7], a modular approach with heuristic tree search [8],

[1]This is the case for the open-source firewalls `pf` [1] and `iptables` [2]. We speculate that this architecture is typical in commercial firewalls as well.

[2]Some firewalls treat UDP traffic as connection-oriented and perform state lookups on UDP packets as well.

[3]packets-per-second.

and two dimensional classification using prefix tuple space and different types of markers [9].

Hash-based algorithms are proposed in [10]. Algorithm uses hash tables for each prefix length and perform a binary search on those hash tables, coupled with various optimizations according to prefix statistics.

However, firewall packet matching uses "first match" semantics, and firewall rules allow arbitrary ranges, not only subnets, in any field. Thus all the above mentioned algorithms are not directly applicable to firewall packet matching. The following papers describe algorithms designed specifically for firewall packet matching.

The algorithm of [11] uses a geometric approach (range queries and interval trees, cf. [12]), implements first-match semantics, and achieves logarithmic time matching, with near-linear space usage and a dynamic data structure that allows fast updates. However, this algorithm works in one dimension, and may be scaled to two dimensions, but it seems hard to extend to more than two dimensions.

The Algorithm in [13] uses a geometric approach with the Area Based Quad-Trees (AQT). It has an $O((\log n)^{d-1})$ time complexity and allows fast updates.

A technique that is similar to ours was used in a recent paper [14], which describes two algorithms: backtracking and set pruning tries. Both perform better than their respective theoretical bounds: $\Omega((\log n)^{d-1})$ time for backtracking and $O(N^d)$ space for set pruning tries. The authors used the field order to reduce the backtracking time, whereas we use the field order to reduce the space required.

1.2. Contributions

In this paper we revisit an algorithm from computational geometry, and apply it to the firewall packet matching. We call this algorithm The Geometric Efficient Matching (GEM) algorithm. This algorithm performs matching in $O(d \log n)$ time, where n is the number of rules in the firewall rule-base and d is the number of fields to match. The worst-case space complexity of GEM is $O(n^d)$. For instance, for TCP and UDP we have $d = 4$, giving a search time of $O(\log n)$ and worst case space complexity of $O(n^4)$.

Our data structure allows easy control over the order of fields to be matched. The data structure can be used for any number of dimensions d, but typical values for firewall packet matching are either $d = 2$ for opaque protocols like IPsec (protocol 50 or 51) or $d = 4$ for TCP, UDP, and ICMP. We focus on the more difficult case for the algorithm, with $d = 4$, in which the match fields are: source IP address, destination IP address, and source and destination port numbers. This fits TCP and UDP filtering, and also ICMP (using the 8-bit message type and code instead of 16-bit port numbers).

The worst case space complexity manifests itself when the rule-base consists of random rules, However, real firewall rule-bases are far from random. Rule-bases collected by the Lumeta Firewall Analyzer [15, 16] show that, e.g., the source port field is rarely specified, and the destination port field is usually a single port number (not a range) taken from a set of some 200 common values.

To evaluate the GEM algorithm beyond a theoretical analysis, we performed an extensive simulation study. We then implemented GEM within the Linux `iptables` open-source firewall [2], and tested its performance in a laboratory testbed.

Based on statistics we gathered from real rule-bases, we created a non-uniform model for random rule-base generation, which we call the Perimeter rule model. On rule-bases generated by this model, we found that the order of field evaluation has a strong impact on the data structure size (several orders of magnitude difference between best and worst). We found that the evaluation order which results in the minimal space complexity is: destination port, source port, destination IP address, source IP address. With this evaluation order, the growth rate of the data structure is nearly linear with the number of rules. The data structure size for rule bases of 5,000 rules is $\approx 13MB$, which is entirely practical. Using more aggressive space optimizations allows us to greatly reduce the data structure at a cost of a factor of 2 or 3 slowdown. For instance, using 3-part heuristic division, we get a data structure size of 2MB for 10,000 rules.

Our `GEM-iptables` Linux implementation sustained a matching rate of over 30,000 pps, with 10,000 rules, without losing packets. In comparison, `iptables` could only sustain a rate of ≈ 3000 pps with the same rule-base.

Thus, we conclude that the GEM algorithm is an excellent, practical, algorithm for firewall packet matching: Its matching speed is far better than the naive linear search, and its space complexity is well within the capabilities of modern hardware even for very large rule-bases.

number	description	space
0	source IP address	32bit
1	destination IP address	32bit
2	source port number	16bit
3	destination port number	16bit
4	protocol	8bit

Table 1. Header field numbering.

2. THE ALGORITHM

The firewall packet matching problem finds the first rule that matches a given packet on one or more fields from its header. Every rule consists of set of ranges $[l_i, r_i]$ for $i = 1, \ldots, d$, where each range corresponds to the i-th field in a packet header. The field values are in $0 \leq l_i, r_i \leq U_i$, where $U_i = 2^{32} - 1$ for IP addresses, $U_i = 65535$ for port numbers, and $U_i = 255$ for ICMP message type or code. Table 1 lists the header fields we use (the port fields can double as the message type and code for ICMP packets).

Remarks:

- We use '$*$' to denote wildcard: An '$*$' in field i means any value in $[0, U_i]$.
- We are ignoring the action part of the rule (e.g., pass or drop), since we are only interested in the matching algorithm.

Let us formally define the point location problem in our context. Each rule can be viewed as a d-dimensional hyper-rectangle, every matching field in rule defines its own dimension. Thus we have n hyper-rectangles, which can overlap. These hyper-rectangles define a d-dimensional space subdivision, which can have an $O(n^d)$ complexity in the worst case. We organize these hyper-rectangles in a point-location data structure, which consists of non-overlapping *simple hyper-rectangles*. Each simple hyper-rectangle is governed by a single rule: This is the first rule that matches all the points (= packets) within the hyper-rectangle. Thus, the packet matching problem becomes a geometric point location problem. The point coordinates are the values of a packet's header fields. Once we identify which hyper-rectangle the point falls into, we know which rule applies and what the decision is for this packet.

The Data Structure: The GEM search data structure consists of two parts. The first part is an array of pointers, one for each protocol number, along with a cell for the '$*$'—all protocols and small header for every cell in array. Header contains information about the *order* of data structure levels and pointer to the first level and the number of simple ranges in that level.

The second part represents the levels of data structure. Every level is a set of nodes, where each node is an array. Each array cell specifies a simple range, and contains a pointer to the next level node. In the last level the simple range information contains the number of the winner rule instead of the pointer to the next level.

The Search Algorithm: The packet header contains the protocol number, source and destination address and port numbers fields. First, we check the protocol field and go to the protocol array of the search data structure, to select the corresponding protocol database header. From this point, we apply a binary search with the corresponding field value on every level, in order to find the matching simple range and continue to the next level. The last level will supply us with the desired result—the matching rule number.

Search time: In each level we execute a binary search on an array of at most $2n$ entries, where n is the maximal number of active rules. We process two searches: one with the packet's protocol and one in the '$*$' data structure. Thus, for d levels, the search time is $O(d \log n)$. For a constant $d = 4$, we get an $O(\log n)$ search time. Note that the '$*$' search data structure only has 2 levels (for IP addresses), thus the search time is dominated by the time to search the 4 levels of the TCP search data structure.

3. THE `GEM-iptables` IMPLEMENTATION

To evaluate GEM in a more realistic environment, we implemented the GEM algorithm and integrated it with the code of the Linux `iptables` firewall. We used Red Hat Linux 9 (kernel version 2.4.18-8) and `iptables` v1.2.8. We incorporated the GEM build algorithm into the user-space program `iptables`, and the GEM search algorithm into the `ip_tables` kernel module. The built GEM database was transferred from user space to the kernel using the mechanism already employed by `iptables`. We left the existing `iptables` linear search algorithm intact. The selection of linear or GEM search was controlled by a command line switch.

Our testbed consisted of two computers, with one acting as the firewall, and the other acting as a packet generator. The firewall was a 2.4GHz Pentium 4 with 512Mb RAM, with two 100Mbps Ethernet interfaces. The packet generator was a 700MHz Pentium III with 396Mb RAM and a single 100Mbps Ethernet interface. Both computers ran Red Hat Linux 9. We connected the two computers by a crossover Ethernet cable. The firewall's `eth1` interface was left unconnected.

3.1. Results and interpretation

We compared the matching throughput of `iptables` and `GEM-iptables` for rule-bases of 2000, 4000, and 10000 rules. For each rule-base size, we varied the packet send rate from 1000pps up to 30,000pps, and recorded the number of received (filtered by `iptables`) packets. The results can be seen in Figure 1. Every point on the curves is an average of 15 runs using three rule-bases of the given size. We also show the 90% confidence intervals.

Figure 1 shows that `iptables` has a maximal throughput of between 2500pps and 9000pps (inversely proportional to the number of rules). This is in line with the results reported in [17] about the matching time of OpenBSD's `pf` [1], versus `iptables` and FreeBSD's `IPFilter` [18]. The reported maximal throughput in [17] was ≈ 2500 pps, for 1000 rules—but the author used a much slower machine than ours.

In contrast, GEM maintained a 100% throughput at all the send rates and for all rule-base sizes we tried. In fact, so far we were unable to reach send rates that cause GEM to lose packets. This is since the packet generating Perl script, running on the slower computer, hit a CPU bottle-

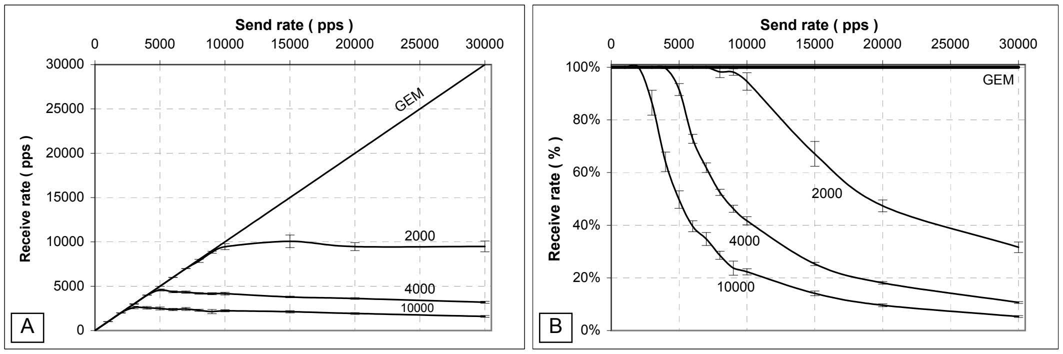

Fig. 1. Throughput of `iptables` with and without GEM, for different rule-base sizes. Figure A shows the receive rate as a function of the send rate, and figure B shows the throughput as a percentage.

neck and could not send more than 30,000pps. Thus we have not determined the maximal throughput of GEM, even with 10,000 rules. Based on the fact that the GEM search time only grows with the log of the number of rules, and on earlier simulation results (omitted), we extrapolate that GEM may well be able to filter at a rate of 100,000pps. This has yet to be verified in the lab.

4. REFERENCES

[1] "PF: OpenBSD packet filter," 2003, `http://www.benzedrine.cx/pf.html`.

[2] "The netfilter/iptables project, v1.2.7," 2002, `http://www.netfilter.org/`.

[3] Anja Feldmann and S. Muthukrishnan, "Tradeoffs for packet classification," in *Proc. IEEE INFOCOM*, 2000, pp. 1193–1202.

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

[5] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," in *Proc. ACM SIGCOMM*, 1999, pp. 135–146.

[6] V. Srinivasan, "A packet classification and filter management system," in *Proc. IEEE INFOCOM*, 2001, pp. 1464–1473.

[7] M. Waldvogel, "Multi-dimensional prefix matching using line search," in *Proceedings of IEEE Local Computer Networks*, Tampa, FL, USA, Nov. 2000, pp. 200–207.

[8] Thomas Y. C. Woo, "A modular approach to packet classification: Algorithms and results," in *Proc. IEEE INFOCOM*, 2000, pp. 1213–1222.

[9] P. R. Warkhede, S. Suri, and G. Varghese, "Fast packet classification for two-dimensional conflict-free filters," in *Proc. IEEE INFOCOM*, 2001, pp. 1434–1443.

[10] V. Srinivasan and G. Varghese, "Faster IP lookups using controlled prefix expansion," in *ACM Conference on Measurement and Modeling of Computer Systems*, 1998, pp. 1–10.

[11] D. Eppstein and S. Muthukrishnan, "Internet packet filter management and rectangle geometry," in *ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2001, pp. 827–835.

[12] M. de Berg, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, 2nd edition, 2000.

[13] M. M. Buddhikot, , S. Suri, and M. Waldvogel, "Space decomposition techniques for fast Layer-4 switching," in *Protocols for High Speed Networks IV*, Aug. 1999, pp. 25–41.

[14] Lili Qiu, G. Varghese, and S. Suri, "Fast firewall implementations for software and hardware-based routers," in *Proc. ACM SIGMETRICS*, 2001.

[15] A. Wool, "Architecting the Lumeta firewall analyzer," in *In Proceedings of the 10th USENIX Security Symposium*, Washington, D.C., August 2001, pp. 85–97.

[16] A. Wool, "A quantitative study of firewall configuration errors," *IEEE Computer*, 2004, To appear.

[17] D. Hartmeier, "Design and performance of the OpenBSD stateful packet filter (pf)," in *Proc. FREENIX Track: 2002 USENIX Annual Technical Conference*, June 2002.

[18] D. Reed, "IP filter," 2003, `http://coombs.anu.edu.au/~avalon/`.