Fast Firewall Implementations for Software and Hardware-based Routers

[Extended Abstract][*]

Lili Qiu
Microsoft Research
liliq@microsoft.com

George Varghese
UCSD
varghese@cs.ucsd.edu

Subhash Suri
Washington University
suri@cs.wustl.edu

ABSTRACT

Routers must perform packet classification at high speeds to efficiently implement functions such as firewalls and diffserv. Classification can be based on an arbitrary number of fields in the packet header. Performing classification quickly on an arbitrary number of fields is known to be difficult, and has poor worst-case complexity.

In this paper, we re-examine two basic mechanisms that have been dismissed in the literature as being too inefficient: backtracking search and set pruning tries. We find using real databases that the time for backtracking search is much better than the worst-case bound; instead of $\Omega((logN)^{k-1})$, the search time is only roughly twice the optimal search time[1]. Similarly, we find that set pruning tries (using a DAG optimization) have much better storage costs than the worst-case bound. We also propose several new techniques to further improve the two basic mechanisms. Our major ideas are (i) backtracking search on a small memory budget, (ii) a novel compression algorithm, (iii) pipelining the search, (iv) the ability to trade-off smoothly between backtracking and set pruning, and (v) algorithms to effectively make use of hardware if hardware is available. We quantify the performance gain of each technique using real databases. We show that on real firewall databases our schemes, with the accompanying optimizations, are close to optimal in time and storage.

1. INTRODUCTION

Routers must perform packet classification at high speeds to efficiently implement functions such as firewalls and diffserv. In packet classification routers, the route and resources allocated to a packet can be determined by the destination address as well as other header fields of the packet such as the source address and TCP/UDP port numbers etc.

Packet classification is performed using a packet classifier, which is a collection of filters (or rules in firewall terminology). Each filter specifies a class of packet headers based on some criterion on K fields of the packet header. Each filter has an associated directive, which specifies how to forward the packet matching this filter. We say that a packet P matches a filter F if each field of P matches the corresponding field of F. This can be either an exact match, prefix match, or a range match. The match type is implicit in the specification of the field. A given packet header can match multiple filters, so each filter is given a cost, and the packet is forwarded using the least cost matching filter.

The key metric is classification speed. It is also important to reduce the size of the data structure to allow it to fit into high speed memory. The time to add or delete filters is often ignored in existing work, but can be important for dynamic filters.

Performing classification quickly on an arbitrary number of fields is a hard problem, and requires $O(N^k)$ memory or $\Omega((\log N)^{k-1})$ search time, where N is the number of filters and k is the number of classified fields [3, 6].

In this paper, we propose several techniques for speeding up packet classification for firewalls using properties we have observed in real firewall databases. [2]

2. EXPERIMENTAL SETTINGS

We experimentally evaluate our techniques by using a set of 5 industrial firewall databases that we obtained from various sites. Table 1 shows the number of rules in the databases. The rules in the firewall specify the following five fields: source/destination IP addresses, source/destination port numbers, and protocol type, where each field is specified either as an exact match, or as a prefix, or as a range.

3. OUR APPROACHES

We re-examine two of the simplest packet classification mechanisms: *backtracking search* and *set pruning tries*. See [5] for

[*]A full version of this paper is available as [5].

[1]The height of the multiplane trie is regarded as the optimal search time throughout the paper, unless otherwise specified.

[2]While we believe our techniques generalize to other filter databases such as diffserv, it is difficult to test this assertion because there are no models of diffserv databases that are generally agreed upon.

Database	# Rules
Database 1	67
Database 2	157
Database 3	183
Database 4	279
Database 5	266

Table 1: Firewall databases.

details. There is an interesting duality between these two simple schemes: backtracking search requires the least storage but can have poor worst-case search times; set pruning tries have minimal search times but have poor worst-case storage. Earlier researchers have dismissed backtracking search as being too slow [7], and dismissed set pruning tries as being suitable only for very small packet classifiers [1, 7].

However, we find that using real databases the time for backtracking search is much better than the worst-case bound. Instead of $(\log N)^{k-1}$, the search time is only a constant factor (often only a factor of two) worse than optimal. Similarly, we find that set pruning tries (using a DAG optimization) have much better storage costs than the worst-case bound of N^k indicates.

We also propose several novel techniques to further improve the performance of backtracking and set pruning tries. First, most designers assume that backtracking is infeasible to implement in hardware because of its high computational state (large number of registers) and its inability to be pipelined (without having memory replicated for each pipeline stage). We show, perhaps surprisingly, that both these assumptions are false. We introduce a new backtracking search technique, which reduces the hardware register cost for doing backtracking in hardware to $D + 1$, where D is the number of dimensions. We also show that while some memory must be replicated at different stages, one can pipeline real firewall databases with a very small amount of replicated memory (much smaller than linear growth with the number of pipelining stages).

Second, we design a novel compression algorithm that applies to any multiplane trie. Our compression scheme is particularly useful because it allows filters that use port ranges to be stored economically. We evaluate our compression scheme using both theory and experiments. Our results indicate that compression reduces the lookup time by a factor of 2 - 5, and reduces the storage by a factor of 4 - 12.

Third, given that backtracking search and set pruning tries are at two ends of a spectrum between optimal storage and optimal time, it makes sense to study the tradeoff between these two extremes. As the two schemes are structurally similar and use multiplane tries as their underlying basis, we show that it is possible to smoothly tradeoff storage for time using a new mechanism called selective pushing. Our results show that the tradeoff scheme offers more choices, and can improve the time of backtracking search with only modest increase in storage.

Finally, we investigate the possibility of moving a subset of filters to hardware if it is available. Our results show that by removing only a small number of filters (10% - 20%) from software for hardware lookup, the storage requirement and query lookup time for the software approach (i.e. backtracking search and set pruning tries) can be greatly reduced. This is significant because they indicate the benefit of adding small ternary CAMs to software-based routers while yet allowing the firewall database to contain a large number of rules.

4. CONCLUSIONS

In this paper, we showed experimentally that the performance of simple trie based filter schemes is much better than worst-case figures predict. We also improved the basic trie based filter schemes with a number of optimizations, including pipelining, compression, selective pushing, and possible hardware support.

Despite the fact that the storage numbers for optimized set pruning tries are reasonable, our final message is that backtracking (with the above optimizations) offers a more reasonable time-space tradeoff than set pruning tries, and should be seriously considered by router implementors.

5. REFERENCES

[1] D. Decasper, Z. Dittia, G. Parulkar, B. Plattner. Router Plugins A Software Architecture for Next Generation Routers. In *Proc. of SIGCOMM*, Sept. 1998.

[2] IETF Differentiated Services Working Group. http://www.ietf.org/html.charters/diffserv-charter.html.

[3] T. V. Lakshman and D. Stidialis. High Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching. In *Proc. of SIGCOMM*, pp. 191-202, Sept. 1998.

[4] P. Gupta and N. McKeown. Packet Classification on Multiple Fields. In *Proc. of SIGCOMM*, pp. 147-160, Sept. 1999.

[5] L. Qiu, G. Varghese, and S. Suri. Fast Implementation for Software-based Router. Cornell University, Computer Science Technical Report, TR2000-1805, July 2000.

[6] V. Srinivasan, S. Suri, and G. Varghese. Packet Classification using Tuple Space Search. In *Proc. of SIGCOMM*, pp. 135 - 146, Sept. 1999.

[7] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast Scalable Level Four Switching. In *Proc. of SIGCOMM*, pp. 203-214, Sept. 1998.