Packet Filtering in High Speed Networks*

Subhash Suri† George Varghese‡

1 Introduction

The commercial viability of the future Internet depends on its ability to provide *differentiated service* to paying customers. The existing Internet delivers only the so-called *best effort* (undifferentiated) service. Examples of differentiated service include firewalls for enterprise networks, bandwidth guarantees to applications, and traffic sensitive routing. These services require Internet routers to move from simple *destination-based packet forwarding* to a more complex form of forwarding called *layer 4 switching*.

In layer 4 switching, each router maintains a database of packet filters, each specifying values for some key header fields. The most common fields are the IP destination address (32 bits), IP source address (32 bits), protocol type (8 bits), and port numbers (16 bits) of destination and source applications. Each field is specified as a variable length prefix. For instance, the filter F = (128.112.*, *, TCP, 20, *) specifies a rule for traffic flow addressed to subnet 128.112 using TCP destination port 20, which is typically used for file transfer; a firewall database may disallow file transfers into its network. (The symbol * denotes the standard wild card character.)

2 The Packet Filtering Problem

Suppose there are K header fields in each packet that are relevant to filtering. Then, each filter F is a K-tuple ($F[1]$, $F[2]$, ..., $F[k]$), with each $F[i]$ a variable length prefix bit string. We say that a packet P matches a filter F if $F[i]$ is a prefix of $P[i]$ for all i, $i = 1, 2, \ldots, K$. (The fields of a packet are fully specified to maximum bit length; for instance, the source and destination of P are full 32 bit addresses.)

Each filter F is associated with an action, denoted $A(F)$, and a packet P matching F should be forwarded according to $A(F)$. Some examples of actions are to forward or drop a packet (firewalls), use a preferred outgoing link (QoS routing), use a preferred output buffer (bandwidth reservation). The filters in general overlap and a packet may match multiple filters. So, we assign each filter a cost, denoted $cost(F)$, and forward the packet using the *least cost filter* matching P. The problem of determining the least cost filter for each incoming packet is called the *packet filtering problem*.

3 Practical Constraints and Related Work

Implementing packet filtering capability in Gigabit routers is critical for providing differentiated service and firewall security. Given that the average size of an IP packets is around 252 bytes, a Gigabit link needs to process about 1–2 million packets per second.

Packet filters can be viewed geometrically as rectangular boxes in a K-dimensional discrete space, and the packet filtering becomes the multidimensional range searching problem, studied in computational geometry [4]. The best data structure for this problem requires $O(N(\log N)^{K-1})$ space, and search time $O((\log N)^{K-1} + L)$, where N is the number of filters and L is the number of filters matching the packet. Unfortunately, these costs are infeasible, even for modest values of N, K. For instance, when $N = 20,000$ and $K = 4$, the worst-case search cost is at least 2916 steps and the memory cost is $2916 \times N$. Even for disjoint (unambiguous) 2-dimensional filters, our experiments show that the optimized point location algorithm of Seidel [5] takes 40 μsecs for 20K filters, which is almost 2 orders of magnitude too slow.

Existing firewall implementations do a linear search, and use caching to improve performance. Even assuming (very optimistically) a 80% hit rate [3, 2], the cost of linear search through $20K$ filters is a bottleneck even if it occurs only 20% of the time.

4 Fast and Scalable Packet Filtering

We briefly describe our ongoing research on the packet filtering problem [6]. A key development is a fast algorithm for 2-dimensional filters—for example, destination-source pair filters. This is an important case used in Virtual Private Networks (VPNs) as well as multicast forwarding. Since the other 3 fields (protocol type and port numbers) are in many cases either fully specified or not at all, one can often use hashing to reduce the general filtering problem to a few instances of the 2-dimensional case. With 20K 2-dimensional filters, our

*Research supported by NSF grant 9628190.

†Department of Computer Science, Washington University, St. Louis, MO 63130. Email: suri@cs.wustl.edu.

‡Department of Computer Science, Washington University, St. Louis, MO 63130. Email: varghese@cs.wustl.edu.

algorithm takes about 0.8 μsecs in worst-case with no caching allowed.

Our algorithm uses hashing, and works in the bit space, exploiting the fact each filter is specified using at most 32 bits. We will describe our algorithm in abstract terms, with the model that each field is specified using at most w bits. Our model assumes perfect hashing, which has been implemented in real systems.

We say that a filter $F = (F[1], F[2])$ belongs to tuple (ℓ_1, ℓ_2) if $F[i]$ is ℓ_i bits long, for $i = 1, 2$. We map each filter $F_1, F_2, \ldots, F_N$ to its unique tuple, and let TS denote the resulting set of tuples. A naive filter matching algorithm is to perform a *lookup* in each tuple (ℓ_1, ℓ_2), which can be done through hashing—concatenate ℓ_1 bits of the packet's destination field with ℓ_2 bits of the packet's source field, and use it as the hash key. While in general, the number of distinct tuples may be small, in the worst-case, the number of hashes required is w^2. Our first result improves this to $2w - 1$ hashes, at the expense of increasing memory by factor w.

THEOREM 4.1. *Let T be a $w \times w$ tuple space, in which tuple (ℓ_1, ℓ_2) contains filters whose two fields are ℓ_1 and ℓ_2 bits long, respectively. Given a packet P, we can determine the least cost filter matching P using at most $2w - 1$ hashes. The scheme requires memory $O(Nw)$.*

Proof. The algorithm uses two key ideas: *precomputation* and *markers*. First, each filter F precomputes the best filter matching it from all the filters stored above it in its column. Second, each filter F leaves an appropriate prefix of itself in all the tuples in its row to its left. With these two operations, we can eliminate either a row or a column after each hash lookup. The total cost is $2w - 1$.

When the tuple space is rectangular, an improved bound is possible. Specifically, if the tuple space is just a line, meaning one of the fields is specified to a fixed number of bits, then one can perform binary search to achieve $O(\log w)$ bound. This suggests an obvious gap between the square and narrow rectangles, which is closed by the following result.

THEOREM 4.2. *Let T be a $r \times c$ rectangular tuple space, with $r \leq c$. Given a packet P, we can determine the least cost filter matching P using at most $2r \log(c/r)$ hashes. The scheme requires memory $O(cN)$.*

Using an adversary argument, we can also show matching lower bounds. We assume that an algorithm can *probe* a tuple, which corresponds to composing a hash key and asking whether there is a filter with that composition. If the probe is *unsuccessful*, then the algorithm can infer that no match exists with any filter in the current row to its right (the information provided by the markers). If the probe is *successful*, then we assume that the algorithm receives full information about *all* the filters whose fields are prefixes of the hash key components (the information determined by precomputation). Within this model, we can prove the following results.

THEOREM 4.3. *One can construct a set of N 2-field filters, mapped to $w \times w$ tuple space, and a packet P, such that determining the best matching filter of P requires at least $2w - 1$ probes of the tuple set. The lower bound can be extended to $r \log(c/r)$ probes in a $r \times c$ tuple space, where $r \leq c \leq w$.*

5 Implementation Results and Discussion

We used a standard database of IP addresses (Mae-East NAP [1]) to generate random filters of valid types: $(D, *, *, *)$, $(D, S, *, *)$, $(D, S, P_1, *)$, (D, S, P_1, P_2), $(D, *, P_1, *)$, $(D, *, P_1, P_2)$; here D, S, P_1, P_2 are destination address, source address, and the two port numbers. For 20K 4-dimensional filters, the algorithm running on a 300 Mhz Pentium Pro required 2 MB of memory and 3.2 μsec worst-case search time. In the case of 20K 2-dimensional filters, the worst-case search time is 0.8 μsec. In particular, the number of memory accesses was 9 in the worst case. More results and some other algorithms are described in [6].

While these numbers are promising, our current research is aimed at reducing worst-case search times further, and to look for more efficient algorithms for the filter problem. Since the lower bounds indicate that highly efficient algorithms are unlikely for the completely general filter problem, we are focusing on the important special cases (such as conflict-free filters, or databases decomposable in few 2-dimensional subsets).

References

[1] Merit Inc. IPMA statistics. (`nic.merit.edu.`)

[2] P. Newman, G. Minshall, and L. Huston. IP Switching and Gigabit Routers. *IEEE Comm. Mag.*, Jan. 1997.

[3] C. Partridge. Locality and route caches. In *NSF Workshop on Internet Statistics Measurement and Analysis*, San Diego, CA, USA, February 1996.

[4] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[5] R. Seidel. A randomized planar point location structure.

[6] V. Srinivasan, G. Varghese, S. Suri and M. Waldvogel. Fast Scalable Algorithms for Level Four Switching. *SIGCOMM '98*, Vancouver.