

O(1) Parallel Lookups for High Speed Layer Four Switching

Giridhar Appaji Nag, Y*
 UbiNetics India Pvt. Ltd.
 Bangalore 560027, India
 (Giridhar.Nag@ubinetics.co.in)

Sajith, G
 Dept. of CSE, IIT Guwahati
 Guwahati 781039, India
 (sajith@iitg.ernet.in)

Abstract—For modern network applications like reservation services, provision of QoS, high speed firewall services etc., packet routing¹ in communication networks requires lookup using higher layer headers (Typically Layer 4 headers in the seven layer ISO/OSI model) in very large routing policy databases. In the current work, the problem of lookups using headers has been abstracted for using parallel algorithmic techniques. The main results are (1) Design of a family of parallel circuits with O(1) lookup time (2) Modification of the circuits to reduce the hardware complexity (such that the lookup time is parameterized by the hardware complexity). The technique presented here does not assume the availability of all the packet headers before starting the processing.

I. INTRODUCTION

One of the basic problems in communication networks is that of sending data from one point (or node of the network) to another. *Routing* is the task of finding the path from a sender to a desired destination. The information required to do this is stored in a *routing table* and the important steps involved are:

- 1) Deciding which routes go into the routing table. This is known as the routing policy and is usually implemented by a user program (for supporting manual insertion of routes, via a routing daemon or ICMP redirect packets).
- 2) Searching the routing table for a matching host (or network) address to decide which interface to deliver the packet on. This is known as the routing mechanism.

Layer 4 switching is different from traditional routing primarily in the routing policy. The routing mechanism may have to be different to accommodate for this change in routing policy.

A. Traditional Routing

Traditionally, the routing information of a layer 3 router (In case of the TCP/IP model, this refers to the IP layer) is organized as a table. A typical routing table would be as shown in Table I.

The second entry in the routing table says that for the destination 202.141.81.3, the gateway to which the packet is to be sent is 202.141.80.2. The flags U, G and H convey the information that the route is **U**p, the packet has to be sent to a **G**ateway and the destination address is a full **H**ost address. The **I**nterface is the physical network on which the packet has to be sent for this particular route and the **M**etric is the distance (or cost) associated with the destination. The

*The current work was submitted in the year 2000 to the Department of Computer Science and Engineering, Indian Institute of Technology, Guwahati.

¹In this paper, we use the terms routing and switching interchangeably

Destination	Gateway	Metric	Flags	Ref	Use	Ifc.
202.141.81.2	*	0	UH	0	0	eth0
202.141.80.3	202.141.80.2	2	UGH	0	0	eth0
202.141.80.2	*	1	U	0	0	eth0
127.0.0.0	*	0	U	0	0	lo0
0.0.0.0	202.141.81.2	1	UH	0	0	eth0

TABLE I
 EXAMPLE OF AN IP ROUTING TABLE

routing decision here is taken only based on the destination address and the metric.

The main disadvantage in a routing table organized as shown in Table I is that, a linear search has to be done to find out the route for a packet and this takes considerable time if there are many entries in the table.

B. Layer 4 Switching

The motivation for the design of a Layer 4 switch is the requirement of being able to provide differentiated services. Provision of QoS, support for reservation protocols, high speed firewall processing and services like VPN (virtual private networks) have to be supported within the same network infrastructure. Layer 4 Switching [1], offers this increased flexibility and service differentiation. For enabling this, a layer four switch would have to look at the packet headers that provide information regarding the application type (source and destination ports), the communication end-points (source and destination IP addresses) and other traffic classifying fields (e.g. the IP-TOS).

The distinguishing factor that can enable this differentiation is the packet classification functionality in a layer 4 router is its capability to look at higher layer headers and screens packets based on them. The fact that this classification is based on multiple fields combined with the volume of rules in the routing table make the problem of lookups interesting.

II. THE PACKET CLASSIFICATION PROBLEM

Traditional routers do a linear lookup on the entries of the routing tables. This works well unless there are a very large number of entries in the routing table to search through. A large routing table is very much the expected situation in a fourth layer switch that would be employed at the edge of networks and in high capacity networks that serve a large pool of users.

A packet P , in the incoming queue of a router is assumed to have k fields $P[1], P[2], \dots, P[k]$. Each $P[j]$ is a string of

bits. The rules for classifying a packet that has to be serviced are known as filters. A filter F_i is a k tuple of headers $H_i[1], H_i[2], \dots, H_i[k]$ (One per header field $P[j]$ of the packet P). A filter database is a set of filters F_1, F_2, \dots, F_n . Each F_i is associated with a cost(F_i).

The problem of packet classification is to determine the best matching filter of lowest cost for each incoming packet at the router. A match could be a perfect match (for applications like firewall processing), or the closest filter that matches the maximum number of header fields (for providing the least expensive outgoing path to a packet).

An example filter in the layer 4 router scenario could be $F = \{SrcIP, DestIP, IP-TOS, Protocol, SrcPort, DestPort, Flags\}$ where $SrcIP$ and $DestIP$ are the source and the destination IP addresses of the packet, $SrcPort$ and $DestPort$ are the respective source and the destination ports, $IP-TOS$ is the Type of Service/Priority filed in the IP header, $Protocol$ is the identifier of the transport layer protocol and $Flags$ are the transport layer flags. The cost function could be the position of the filter in the filter table [the cost function in a traditional layer 3 routing table (Table I) would be the metric field].

III. RELATED WORK

Prior work in packet classification employs a variety of techniques and algorithms like the Grid of Tries and Cross Producing [3]. Grid of Tries is a good technique to use if the filtering is based only on the source and destination address (Or any two) fields of the packet headers. For multi-dimensional filters, the paper describes another technique called Cross Producing. The basic idea in Cross Producing is to pre-compute the best matching filter for all possible packet combinations. Few optimizations are discussed to reduce the memory requirements of both these techniques.

A generic packet classification algorithm known as Tuple Space Search that exploits the fact that real filter databases use only a small number of distinct field lengths is described in [5]. In this technique, packet filters are mapped to tuples which are maintained as hash tables that can be searched in one memory access.

Multi-Dimensional Range Matching [4]. is one scheme that is optimized for implementation in hardware and also uses bit parallelism to match multiple fields in parallel. For the special case of classification in two dimensions, the paper describes techniques for achieving higher performance.

A heuristic algorithm called Recursive Flow Classification (See [2].) exploits the fact that classifier databases contain lot of redundant information and are also well structured.

We introduce a new data structure called as 2^b Trie which is a modification of the Trie used in the Grid of Tries algorithm. The observation that there are few distinct header field values from the filter database is also used in our algorithm, but we do not pre-compute all the possible combinations of filters that could arise from them. Instead, 2^b

Tries are created out of these distinct header fields. However, we do not arrange the source and destination Tries as a Grid. The consequence of this is that the proposed scheme is extensible to multiple packet header fields and unlike the Grid of Tries algorithm, it is not restricted to two dimensional filters.

IV. PROPOSED ALGORITHM - DELAY ACCOMMODATING 2^b TRIES

Existing schemes for packet filtering assume that the headers of the packet for which the lookup has to be performed are all available. This is a costly assumption if the packets are arriving at a very high rate (which is usually the case in optical networks). The proposed schemes are geared at hardware implementations assuming reasonably static data structures. Parallelism is also exploited. The difficulty in using parallel algorithmic techniques has to be appreciated, as, we are not interested in waiting for all the bits of the packet to arrive but would like to process them at wire speed (due to this, the problem is inherently sequential).

A. Key Ideas

The key observations that lead to the 2^b Tries scheme are as follows:

- 1) An incoming stream of bits from a packet can be passed through the 2^b Tries and all the filters that match a particular header field can be found in constant time.
- 2) For a given stream of input bits of a packet, a table with the information about all the matching filters (per header field) would be available after all the required bits of the packet header have been received.
- 3) Using a grid of hardware comparators, the constructed table can be used to find out the best matching filter.

B. The Algorithm

1) *Filter Table*: Consider the filter database in Table II with three fields that are used for classifying a packet. The filters are assumed to be ordered with the least cost filter first.

Filter	H_1	H_2	H_3
F_1	100*	0111*	001*
F_2	0001*	100*	1100*
F_3	100*	0110*	001*
F_4	0010*	110*	0111*
F_5	1011*	0100*	1100*
F_6	100*	110*	001*
F_7	0010*	0110*	0111*
F_8	0001*	0100*	001*
F_9	110*	110*	1100*
F_{10}	111*	0111*	1100*
F_{11}	110*	0111*	0111*
F_{def}	default	default	default

TABLE II
AN EXAMPLE THREE FIELD FILTER DATABASE

The first step is to collect distinct masks from each field. The result would be as shown in Table III.

H_1	H_2	H_3
100*	0111*	001*
0001*	110*	1100*
0010*	0110*	0111*
1011*	0100*	
110*		
111*		

TABLE III
THE SLICED FILTER DATABASE

2) *Construction of 2^b Tries:* A new data structure called as 2^b Trie is proposed for use in filter matching. A 2^b Trie is a 2^b -ary tree with each branch labeled with a unique value in the range 0 to $2^b - 1$. The value of b should be adjusted such that the time taken to process b bits is nearly the same as the time taken to receive b bits in the incoming path of the router. The choice of b is important if the speed of incoming data is very high and the time taken to process the input bits is more than the time taken for the bits to arrive.

One 2^k Trie (Call this T_i) is constructed per header field H_i as follows:

- Each branch of the Trie T_i is associated with a k bit number that is a part of H_i .
- Traversing down a path of T_i leads us to one of the members in the distinct elements list of the header field H_i .
- The leaves of T_i have the pointers to the filters that have their field mask as the one that would be obtained by traversing along the path from the root to the the leaf of T_i .

If the length of field entry H_i is not an integral multiple of k , it may be converted into two or more by breaking it into multiple masks. For example, 100* may be written as (1000* or 1001*). In the above example, the Tries (with $k = 2$) would be as shown in Figures 1, 2 and 3.

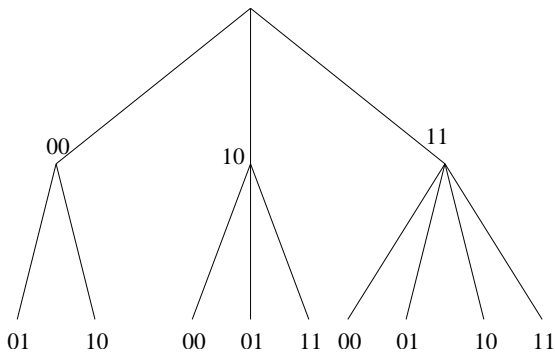


Fig. 1. Trie T1 for Field 1: 100*, 0001*, 0010*, 1011*, 110* and 111*

3) *Finding Matching Fields:* When a new packet (say P) arrives at the input path of the router, the bit stream is propagated down the Tries and the bits in the i^{th} packet header $P[i]$ are compared with the bit labels at the branches of the Trie T_i (corresponding to the header field H_i) and the

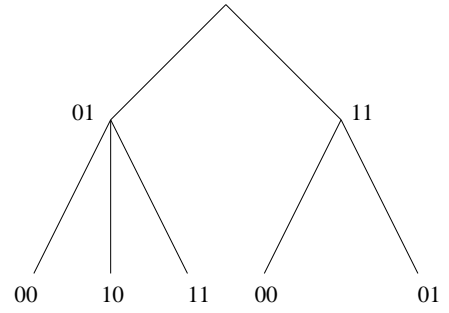


Fig. 2. Trie T2 for Field 2: 0111*, 110*, 0110* and 0100*

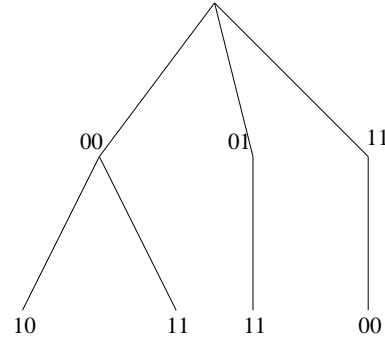


Fig. 3. Trie T3 for Field 3: 001*, 1100*, and 0111*

packet is propagated down the matching branch. The number of phases of propagation would be the same as the depth of the trie. (e.g. The various phases, for a header field $P[2] = [0110^*]$ being propagated down the Trie T_2 would be as shown in Figure 4 and 5).

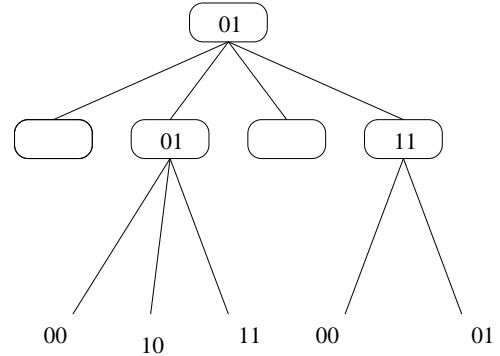


Fig. 4. First phase in the packet header filed [0101*] being propagated down the Trie T_2

At each phase, The matching branch (from among the 2^k branches at that level) is searched for the current k bits of the packet header $P[i]$. Parallel comparator elements (see Figure 6) are used to do the comparison, hence the comparison can be done in constant time. The comparator element circuit is then propagated down the matching branch. This comparative propagation will specify all the filters that have a H_i header field that matches the header field $P[i]$ of the current packet

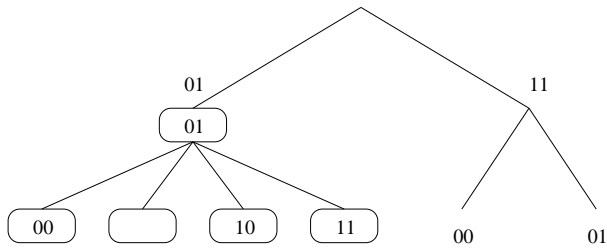


Fig. 5. Second phase in the packet header filed [0101*] being propagated down the Trie T_2

P . If the speed of the incoming data is very high, and the processing takes time equal to that of arrival of b bits, using a 2^b Trie would be optimal (as the input stream of bits would not have to wait for the comparison service).

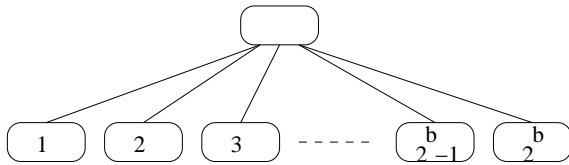


Fig. 6. Parallel comparator network. The root is compared with each of the leaves

A bitmap for the filter table records all the filters that match a particular header field H_i . The number of matching headers for each of the filters fields are stored in a register **Sum**. The bitmap in this case would be will be as shown in Table IV. For each filter, we assume the presence of a hardware element that can set or reset the bits in its bitmap.

After the traversal down each Trie, all the filters in which the current packet header field $P[i]$ is matched by the corresponding field H_i are known. In parallel, the bits in the bitmap corresponding to these fields in the filters are set and the **Sum** values are incremented. If the packet were $P = [100^*][1100^*][001^*]$ then the bitmap and the **Sum** array would be as shown in the Table IV. After all the packet header entries $P[i]$ of the packet P have been matched, it remains to find out the first filter in the table which has a maximum number (or all) of matching headers.

4) Best Matching Filter:

a) *Perfect Match*: A Filter F would be a perfect match for a packet P if each $P[i]$ matches the corresponding H_i from F . The first among the perfect matching filters is the best matching filter with the least cost.

As soon as the last header is matched in the packet, the **AND** of all the bits corresponding to the results in the bitmap for each of the filters is found (Let this be called as the *Boolean And Score* or the **BAS** of the filter). All the filters that have a **BAS** of 1 are a perfect match for that particular packet.

The least cost filter among many perfect matches can be

Filter	H_1	H_2	H_3	Sum	IsMax
F_1	1	0	1	2	1
F_2	0	0	0	0	0
F_3	1	0	1	2	1
F_4	0	0	0	0	0
F_5	0	1	0	1	0
F_6	1	0	1	2	1
F_7	0	0	0	0	0
F_8	0	1	1	2	1
F_9	0	0	0	0	0
F_{10}	0	0	0	0	0
F_{11}	0	0	0	0	0
F_{def}	1	1	1	3	NA

TABLE IV
BITMAPS AND THE **Sum** ARRAY **Sums** WITH THE PACKET P
= $[100^*][1100^*][001^*]$

found as follows:

At each of the j^{th} filter that has its **BAS** as 1, do an **OR** on the first j **BAS** values. The **ORing** is done in parallel at all the filters that have a **BAS** score as 1 and the only filter with the result of the **OR** as 1 but with its previous filter **OR** result as 0 would be output (See [6]., Chapter 10). Assuming that the **OR** and **AND** operations are constant time, the perfect matching least cost filter can be found in constant time (infact, the result is available the moment all the bits of the packet header have been received). In our example, only the default filter F_{def} satisfies the perfect match criterion.

b) *Nearest Match*: All filters F , that have the maximum value of the **Sum** are the closest matching filters. The first among these is the least cost and nearest matching filter.

The least cost filter among all the nearest matches can be found as follow:

For N filters, using N^2 processor elements (a $N \times N$ grid), a local flag at node (i, j) in the grid is set if $Sum_i \geq Sum_j$. Let $IsMax_i$ be the binary **AND** of all the flags in a row i . Note that $IsMax_i = \text{bool}(\text{IsMax}(Sum_i) == \text{TRUE})$. This operation sets a bit for all the filters that have the maximum value of **Sum**. An example of this operation is illustrated in Table V. It is a constant time operation (See [6]., Chapter 10) to find out the first one among these that has the $IsMax_i$ value as 1.

	Filter	1	2	3	4	5
		Sum_1	Sum_2	Sum_3	Sum_4	Sum_5
	\geq	0	3	2	3	1
Sum_1	0	1	1	1	1	1
Sum_2	0	0	1	0	1	0
Sum_3	0	0	1	1	1	0
Sum_4	0	0	1	0	1	0
Sum_5	0	0	1	1	1	1
	$IsMax$	0	1	0	1	0

TABLE V
EXAMPLE: FINDING ALL THE FILTERS THAT HAVE A MAXIMUM VALUE OF Sum IN AN ARRAY $Sums = \{0, 3, 2, 3, 1\}$

In our filter database, the final **Sum** array would be **Sums** = {2, 0, 2, 0, 1, 2, 0, 2, 0, 0, 0}, the closest matching filters are F_1 , F_3 , F_6 and F_8 and the least cost filter among these would be F_1 (See Table IV).

V. COMPLEXITY

A. Algorithm and Hardware Complexity

The Delay Accommodating 2^b Tries algorithm involves the following steps:

- 1) Passing the incoming stream of bits from a packet through 2^b Tries. All the filters that match a particular header field can be found in $O(1)$ time.
- 2) Computing the **BAS** and finding out the packets that have a **BAS** as 1 (all the perfect matches) or the maximum sum (the best matches), is an $O(1)$ operation.
- 3) Finding the first among the best or the perfect matches can be achieved in $O(1)$ time.

The maximum number of processor elements used in any of the above steps is $O(N^2)$.

Hence the hardware complexity of the algorithm is $O(N^2)$ and the time complexity is $O(1)$.

B. Reducing Hardware Complexity

Though the time complexity of the above algorithmic is $O(1)$, the hardware complexity is $O(N^2)$ (Quadratic in the number of filters). In order to reduce this, the **Sums** array can be broken into parts of size P each. By using $\frac{N}{P}$ smaller grids of size $P \times P$, the $\frac{N}{P}$ best matching filters can be found in constant time. Further matching within this reduced set can be done recursively by reducing the number of filters to $\frac{N}{P^2}$, $\frac{N}{P^3}$, ... $\frac{N}{P^t}$. For each iteration, the number of matching filters is reduced by P times. The hardware complexity is $O(P^2)$ with the lookup time being t where t is the largest integer such that $\frac{N}{P^t} > 1$. That is $O(\log_P N)$.

VI. CONCLUDING REMARKS

The current work examines the problem of packet classification from a parallel algorithmic standpoint. Lookups for packet classification has a lot of importance in high speed communication networks and are known to consume considerable amount of time and memory. We have presented a new algorithm that uses Delay Accommodating 2^b Tries to solve this problem. The parallel circuits used take a constant amount of time for lookup and processes packets as they arrive. The algorithm can be tuned to the speed of the incoming traffic to perform optimally. Even though the hardware complexity of the technique is high, it can be reduced by an increase in the time (parametrized by the hardware complexity) complexity. Future work in this area could involve implementing the above algorithms in a hardware design language and comparing the performance against existing schemes.

VII. ACKNOWLEDGEMENTS

The authors gratefully acknowledge the comments of the anonymous reviewers and several other people for useful inputs via discussions.

VIII. REFERENCES

- [1] J. McQuillan, "Layer Four Switching", Data Communications. October 21, 1997
- [2] Pankaj Gupta, Nick McKeown, "Packet Classification on Multiple Fields", Proceedings of the ACM SIGCOMM 99, September 1999
- [3] V Srinivasan, G Varghese, S Suri, M Valdvogel, "Fast and Scalable Layer Four Switching", Proceedings of the ACM SIGCOMM 98, October 1998
- [4] T V Lakshman, D Stiliadis, "High Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching", Proceedings of the ACM SIGCOMM 98, October 1998
- [5] V Srinivasan, S Suri, G Varghese, "Packet Classification using Tuple Space Search", Proceedings of the ACM SIGCOMM 99, September 1999
- [6] Joseph Jaja, "Parallel Algorithms", Addison Wesley Inc., 1992