Application of Logic Minimization to IP Routing Table Compression

Jacob Bachmeyer

Sunil P Khatri

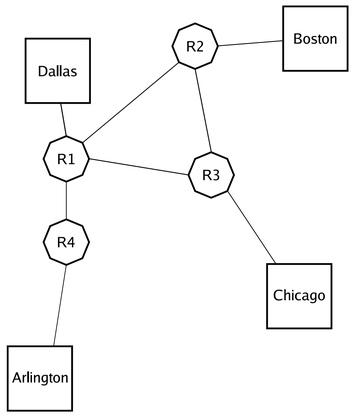
Abstract

The Internet has continued to grow at a dramatic rate. This growth has brought with it a steady increase in the size and complexity of routing tables. We propose a scheme to reduce the complexity of the hardware required to provide fast lookup into these tables by simplifying the table structure. Our approach also reduces the size of the routing table.

We use the well-known Espresso-MV logic minimization algorithm to reduce the size of the routing table, by considering the table as an incompletely specified multi-valued logic function. We have also tried various means of keeping the running time of our algorithm down in the face of ever-larger routing tables. This leads to a trade-off between the size of the produced table and the time required to compress the table.

We also propose a fast update algorithm for the reduced table that does not require further use of Espresso.

A Basic Description of Internetwork Routing



The Internet does not require each host to know how to reach another host, only the address of the host to reach. The network routers find a path by which to deliver each packet.

Each router must maintain a table indicating where to send packets with various destination addresses. This is the *routing table* and its size is dependent on the complexity of the network.

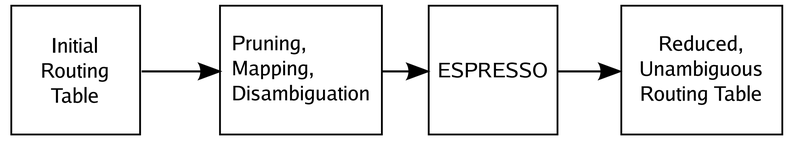
We propose a means to reduce the size of these tables and to reduce the complexity of looking up entries.

Ambiguity and Longest Prefix Matching

128.32.0.0	/16	1
128.32.1.0	/24	3
128.32.1.128	/25	2
128.32.2.0	/24	2
129.0.0.0	/8	1

This is a sample routing table.
Note that the destinations for most of these prefixes are at first glance ambiguous. Does a packet sent to 128.32.1.130 go through port 2, 3, or 1? This is resolved by using only the match with the the longest prefix, so for this example, a packet sent to 128.32.1.130 would be forwarded through port 2.

Overview of our process



Pruning

Frequently in real routing tables, the longest prefix for a given address is sent to the same interface as one of the shorter prefixes.

In such cases, a trivial means of reducing the table is to simply drop the unneeded longer prefixes.

In this example, the route for 128.32.2/24 going to port 2 can be dropped.

128.32.0.0	/16	2
128.32.1.0	/24	3
128.32.1.128	/25	2
128.32.2.0	/24	2
129.0.0.0	/8	1

↓

128.32.0.0	/16	2
128.32.1.0	/24	3
128.32.1.128	/25	2
129.0.0.0	/8	1

Mapping Routes into MV cubes

We simply represent the prefix in binary, with one literal per bit in an address and assign the cube an output value equal to its port number. This gives a form acceptable for ESPRESSO.

127/8	2	→	01111111---------- 2

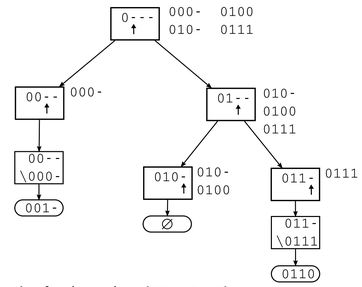
Disambiguation

Since ESPRESSO has no concept of "longest prefix", we can't rely on longest prefix matching if we want to combine routes with different prefix lengths. We must break up large cubes so that they do not intersect any smaller cubes contained in them that correspond to routes using a different port.

For this we use an algorithm we call *split*.

Split

Split is easily explained by example:
Here we calculate 0--- \ {000-,010-,0100,0111}



Thus, the final result is {001-,0110}.

For incremental updates, we generalize this algorithm to allow don't-care terms to appear anywhere in any cube.

Reducing the table using ESPRESSO

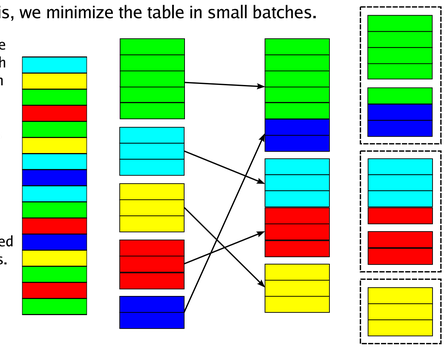
The simple approach of simply giving ESPRESSO the entire table in one piece results in an unnacceptably long running time.

To avoid this, we minimize the table in small batches.

This is an example of this method with 3 bins and a batch size of 4.

Each color stands for a different output port.

We found that 4 bins and a batch size of 2000 worked best on real tables.

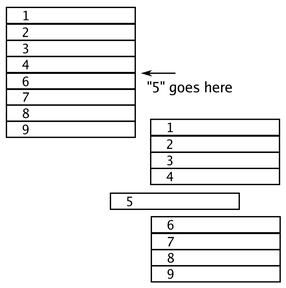


Forwarding Table Updates

To update the forwarding table, we do a binary search to locate where the affected entry should be, then, for insertions, replacing the entry if found or inserting it (and sliding the others down) if not. For removals, we simply slide all entries after the affected up if found, or do nothing if not found.

As an example, we will add an entry that sorts as "5" to the table above.

To simplify the illustration, the prefixes have been replaced with numbers representing their sort order.



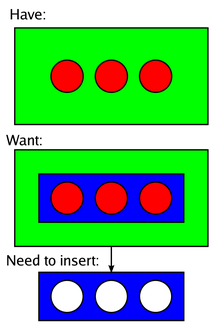
Active Table Updates

For the active table, updates are applied by performing logic operations on cubes.

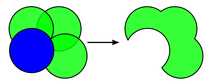
Unlike the forwarding table, which is kept sorted, the active table must be searched cube-by-cube for each update.

Applying any update to the active table is performed by first "cutting a hole" and then possibly "filling it in".

As an example, we will add the blue annulus, not stepping on the red cubes it contains.



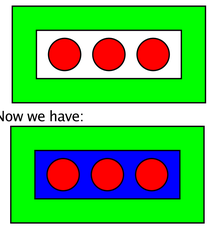
We then find all cubes in the active table that intersect this annulus. In this example, they will all be green.



We replace each such intersecting cube with the difference of itself and the cover we are changing.

We repeat this for each cube in the annulus.

This gives us a table with a hole where those cubes were. For removals, we are done, for insertions, we then append the annulus to the table.

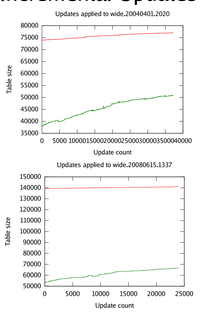


Experimental Results

Table Reduction

Routing Table	#IF	Initial Size	%-reduced	Runtime (sec)
eqix.20061216.2016	66	203782	49.6	135.4
eqix.20080412.1235	10	252209	52.3	167.7
linx.20040601.1227	231	140881	26.7	129.1
linx.20050228.2012	269	157630	19.5	150.8
linx.20070816.1124	402	229884	28.3	206.0
paix.20050617.0927	81	165827	13.4	195.0
paix.20070603.2354	77	225603	23.5	246.1
paix.20080412.1651	80	257034	36.7	253.6
wide.20040401.2020	57	136762	72.3	64.6
wide.20060926.0040	59	199899	52.9	135.1
wide.20070212.2004	49	211622	79.6	83.8
wide.20080615.1337	46	259930	79.5	105.9

Incremental Updates



Department of Electrical and Computer Engineering
Texas A&M University
College Station, TX 77843-3128