EE384y Project Proposal

Katerina Argyraki, Jim Washburn

April 16, 2003

Contents

1 Main Ideas

This project is in the area of network lookup algorithms. More specifically, in the area of optimizing lookup times by taking into account the relative frequency of access of the keys.

There is previous work in this area [1] [2]. Like the previous work we will assume that frequency of accesses to the keys is not uniform, and in fact attempt to take advantage of this fact to improve lookup performance. This project would differ from previous work in that it would not assume that the relative frequencies of access are known a priori. It would be worthwhile to remove this assumption for the following reasons:

- In a networking environment, where we are doing lookups on IP addresses or MPLS tags for example, such relative frequency information does not typically exist, or at least is not knowable to software which builds the data structures to be searched . The basic reason it does not exist is that the keys are typically added to the table incrementally and continuously, interspersed in time with table searches. In other words

we do not have a model in which a set of keys are given and compiled, and searches happen after that point.

- Even if we did have such information (in the form of a counter per key for example) it would be expensive in terms of time to access and maintain the information, and would add to memory consumption.

So consider the following alternative approach. What we will try to do is minimize the expected value of the access time to a data structure, by continuously and incrementally readjusting it. And we'll do this at the time of access to the data structure. We'll use and develop randomized algorithms to do this. As perhaps the simplest example, for a hash table with chaining (linked list), when we access a key, with some probability $p \ll 1$, we will move the key 1 place towards the head of the linked list. In another basic example, in a binary search tree, when we match a key, with $p \ll 1$ we can do a BST "rotation" operation to move the node 1 place upwards towards the root of the tree.

The idea of optimizing the lookup structure by computations at the time of access has not been suggested before in a networking context as far as I know. The many papers written in recent years proposing IP lookup algorithms seem to have the basic paradigm of compiling an optimized data structure once and for all, then doing searches on it henceforth. The reason for the optimization/search dichotomy is probably the underlying assumption that optimization is very complex and slow (so should be done in software) and searches must be simple and fast (so should be done in specialized hardware.) However with the advent of network processors in recent years, this hardware/software split increasingly does not apply. That is, in many systems there is only one processor (or array of identical processors) performing both search and table building operations. Also, if a randomized, amortized optimization algorithm can be made fast enough to execute, and in any case only executed with $p \ll 1$ per search, the other part of that assumption also no longer holds.

Looking in the literature, I find that the idea of adjusting lookup structures at search time has some previous work: [3] [4] [5] . But I do not believe that anyone has explored this in the context of network lookup algorithms.

2 Schedule/Timetable of work

This is a suggested initial plan of action.

- Learn PALAC, the packet classification analysis/design/comparison software package at Stanford.
- Modify PALAC to add the following capability (if it doesn't already have it): Be able to create sets of keys with different probability distributions. Would be also good if the distribution can change over time.
- Consider first IP lookups done as intervals in a BST. The BST data structure has the advantage here that incremental changes can be made locally to the data structure without affecting the rest of the tree. First, be able to build a standard (randomized) BST. That is, add this capability to PALAC if it doesn't already have it. Then add the capability for 1) single rotation 2) move accessed node to root. For both 1) and 2), see how we do with respect to different key prob. distributions in comparison with other lookup structures in PALAC. Are there other possibilities besides 1) and 2) for incremental changes here?
- Think about other data lookup structures such as tries and LULEA. It seems at first glance that the idea of randomized tree optimization on access would not work as well for tries. If for example if we collapse several nodes to shorten the path to a key, this would affect other keys as well. But consider it at least.
- Look at whether this approach can have some benefit for existing multifield classification algorithms. Whenever a linear list of rules occurs, such as at the leaves in HiCuts [7] or other classification data structures [6] [9], it seems clear that randomized move-to-front within the list could be expected to help somewhat. Quantify how much improvement in the number of memory accesses could be expected. One thing to watch out for here is that the list of rules has an implicit priority to it. That is, when multiple rules match, the first rule listed originally input by the user is the one intended to be selected. So we can't move rules around in the list arbitrarily. But the situation isn't as bad as

one might think, because the restriction only applies when the rules "overlap", i.e. match a certain class of inputs in common. So we can move one rule vs. another in a list as long as we check that the two do not overlap. This is a well-known procedure [8]

Also, if the multifield data structure is any sort of tree structure such as in papers mentioned above, then reorganizing the tree itself might be possible based on access. This may be quite complicated, how complicated this would be to do would depend on the data structure.

- Again considering multifield, is there a possibility to use a BST here, where the key of each node is an N-dimensional region. (N=number of fields). The ruleset would have to be preprocessed into a set of non-overlapping regions. The "less-than" operation (necessary for the BST) would be based on the distance of the midpoint of the region from the origin. It would appear that we can define such a BST. If so that would definitely allow the same sort of tree rotation operations as for the single-field case. The basic access time would then be $log(numberofregions)$, with some improvement from there based on the self-adjusting heuristics.

3 Deliverables

- Simulation results
- Optimization Algorithm(s)
- Algorithm Analysis

References

[1] Pankaj Gupta et. al., *Near-Optimal Routing Lookups with Bounded Worst Case Performance ,* , [INFOCOMM]

[2] G. Cheung et. al., *Optimal Routing Table Design for IP Address Lookup under Memory Constraints* , [INFOCOMM 1999]

[3] C. Martinez, S. Roura, *Randomized Binary Search Trees*, [1997]

[4] D. Sleator, R. Tarjan, *Self-Adjusting Binary Search Trees* , [ACM, 1985]

[5] B. Allen, I. Munro *Self-Organizing Binary Search Trees*, [ACM Journal, 1978]

[6] F. Baboescu, et. al. *Packet Classification for Core Routers, Is there an alternative to CAMs?*, [Proc. INFOCOMM 2003]

[7] P. Gupta, N. McKeown. *Packet Classification using hierarchical intelligent cuttings*, [Proc. Hot Interconnects VII, Aug. 1999]

[8] A. Hari, et al, *Detecting and Resolving Packet Filter Conflicts* , [Proc. INFOCOMM 2000]

[9] T. Woo, *A Modular approach to Packet Classification*, [Proc. INFOCOMM 2000]