Packet Classification Using Pipelined Multibit Tries *

Wencheng Lu and Sartaj Sahni
Department of Computer and Information Science and Engineering,
University of Florida, Gainesville, FL 32611
{wlu, sahni}@cise.ufl.edu

August 16, 2006

Abstract

We propose heuristics for the construction of variable-stride one-dimensional as well as fixed- and variable-stride two-dimensional multibit tries. These multibit tries are suitable for the classification of Internet packets using a pipelined architecture. The variable-stride one-dimensional tries constructed by our heuristic require significantly less per-stage memory than required by optimal pipelined fixed-stride tries. Also, the pipelined two-dimensional multibit tries constructed by our proposed heuristics are superior, for pipelined architectures, to two-dimensional multibit tries constructed by the best algorithms proposed for non-pipelined architectures.

Keywords
Packet classification, longest matching prefix, controlled prefix expansion, fixed-stride tries, variable-stride tries, two-dimensional tries, dynamic programming.

1 Introduction

Internet packets are classified into flows based on their header fields. This classification is done using a table of rules in which each each rule is a pair (F, A), where F is a filter and A is an action. If an incoming packet matches a filter in the rule table, the associated action specifies what is to be done with this packet. Typical actions include packet forwarding and dropping. A d-dimensional filter F is a d-tuple $(F[1], F[2], \cdots, F[d])$, where $F[i]$ is a range that specifies destination addresses, source addresses, port numbers, protocol types, TCP flags, etc. A packet is said to match filter F, if its header field values fall in the ranges $F[1], \cdots, F[d]$. Since it is possible for a packet to match more than one of the filters in a classifier, a tie breaker is used to determine a unique matching filter.

In one-dimensional packet classification (i.e., $d = 1$), $F[1]$ is usually specified as a destination address prefix and lookup involves finding the longest prefix that matches the packet's destination address. Data structures for longest-prefix matching have been extensively studied (see [24, 26], for surveys). Although one-dimensional prefix filters are adequate for destination based packet forwarding, higher dimensional filters are required for firewall, quality of service, and virtual private network applications, for example. Two-dimensional prefix filters, for example, may be used "to represent host to host or network to network or IP multicast flows" [12] and higher dimensional filters are required if these flows are to be represented "with greater granularity." Eppstein and Muthukrishnan [7] state that "Some proposals are underway to specify many fields ... while others are underway which seem to preclude

*This research was supported, in part, by the National Science Foundation under grant ITR-0326155

using more than just the source and destination IP addresses ... (in IPsec for example, the source or destination port numbers may not be revealed)." Kaufman et al. [15] also point out that in IPsec, for security reasons, fields other than the source and destination address may not be available to a classifier. *Thus two-dimensional prefix filters represent an important special case of multi-dimensional packet classification.* Data structures for multi-dimensional (i.e., $d > 1$) packet classification are developed in [1, 2, 3, 6, 7, 8, 9, 12, 17, 22, 28, 29, 31, 10, 11], for example.

Srinivasan and Varghese [31] proposed using two-dimensional one-bit tries for destination-source prefix filters. Srinivasan and Varghese [31] also propose extensions to higher-dimensional one-bit tries that may be used with d-dimensional, $d > 2$, filters. Baboescu et al. [3] suggest the use of two-dimensional one-bit tries with buckets for d-dimensional, $d > 2$, classifiers. Basically, the destination and source fields of the filters are used to construct a two-dimesional one-bit trie. Filters that have the same destination and source fields are considered to be equivalent. Equivalent filters are stored in a bucket that may be searched serially. Baboescu et al. [3] report that this scheme is expected to work well in practice because the bucket size tends to be small. They note also that switch pointers may not be used in conjunction with the bucketing scheme. Lu and Sahni [19] have developed fast algorithms to construct space-optimal constrained two-dimensional multibit tries for Internet packet classifier applications.

In this paper, we focus on the development of data structures suitable for ASIC-based pipelined architectures for high speed packet classification. Basu and Narlikar [4] and Kim and Sahni [16] have proposed algorithms for the construction of optimal fixed-stride tries for one-dimensional prefix tables; these fixed-stride tries are optimized for pipelined architectures. Basu and Narliker [4] list three constraints for optimal pipelined fixed-stride multibit tries:

C1: Each level in the fixed-stride trie must fit in a single pipeline stage.

C2: The maximum memory allocated to a stage (over all stages) is minimized.

C3: The total memory used is minimized subject to the first two constraints.

Basu and Narliker [4] assert that constraint C3 reduces pipeline disruption resulting from rule-table updates. Although the algorithm proposed in [4] constructs fixed-stride tries that satisfy constraints C1 and C2, the constructed tries may violate constraint C3. Kim and Sahni [16] have developed faster algorithms to construct pipelined fixed-stride tries; their tries satisfy all three of the constraints C1–C3. FSTs that satisfy C1–C3 are called *optimal pipelined FSTs*.

In this paper, we propose heuristics for the construction of pipelined variable-stride one-dimensional as well as pipelined fixed- and variable-stride two-dimensional multibit tries. The pipelined tries constructed by our algorithms are compared, experimentally, to those constructed by the algorithms of Kim and Sahni [16] and Lu and Sahni [19]. The variable-stride one-dimensional tries constructed by our heuristic require significantly less per-stage memory than required by optimal pipelined fixed-stride tries. Also, the pipelined two-dimensional multibit tries constructed by our proposed heuristics are superior, for pipelined architectures, to two-dimensional multibit

tries constructed by the best algorithms proposed in [19] for non-pipelined architectures.

We begin, in Section 2, by reviewing basic concepts related to the trie data structure. This section also describes multibit fixed- and variable-stride tries, prefix expansion [30], and two-dimensional 1- and multi-bit tries [19]. In Section 3 we develop an (heuristic) algorithm for pipelined one-dimensional variable-stride tries. For two-dimensional tries, we consider two strategies to map a two-dimensional trie onto a pipelined architecture– course grain and fine grain. Our algorithms for coarse-grain mapping are developed in Section 4 and those for fine-grain mapping in Section 5. An experimental evaluation of our algorithms is conducted in Section 6.

2 Tries

2.1 One-dimensional 1-bit Tries

A one-dimensional *1-bit trie* is a binary tree-like structure in which each node has two element fields, *le* (left element) and *re* (right element) and each element field has the components *child* and *data*. Branching is done based on the bits in the search key. A left-element child branch is followed at a node at level i (the root is at level 0) if the ith bit of the search key is 0; otherwise a right-element child branch is followed. Level i nodes store prefixes whose length is $i+1$ in their data fields. A prefix that ends in 0 is stored as $le.data$ and one whose last bit is a 1 is stored as $re.data$. The node in which a prefix is to be stored is determined by doing a search using that prefix as key. Let N be a node in a 1-bit trie and let E be an element field (either left or right) of N. Let $Q(E)$ be the bit string defined by the path from the root to N followed by a 0 in case E is a left element field and 1 otherwise. $Q(E)$ is the prefix that corresponds to E. $Q(E)$ is stored in $E.data$ in case $Q(E)$ is one of the prefixes to be stored in the trie.

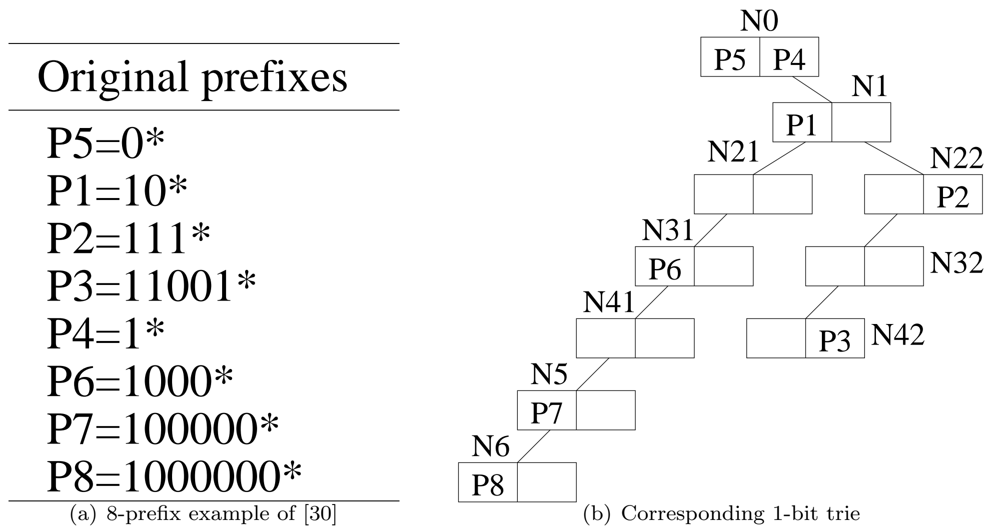
Figure 1 shows a set of 8 prefixes and the corresponding 1-bit trie. The $*$ shown at the right end of each prefix is used neither for the branching described above nor in the length computation. So, the length of $P1$ is 2.

2.2 One-dimensional Multibit Tries

The *stride* of a node is defined to be the number of bits used at that node to determine which branch to take. A node whose stride is s has 2^s element fields (corresponding to the 2^s possible values for the s bits that are used). Each element field has a data and a child component. A node whose stride is s requires 2^s memory units (one memory unit being large enough to accomodate an element field). Note that the stride of every node in a 1-bit trie is 1.

In a *fixed-stride trie* (FST), all nodes at the same level have the same stride; nodes at different levels may have different strides. In a *variable-stride trie* (VST), nodes may have different strides regardless of their level.

Suppose we wish to represent the prefixes of Figure 1(a) using an FST that has three levels. Assume that the strides are 2, 3, and 2. The root of the trie stores prefixes whose length is 2; the level one nodes store prefixes whose length is 5 (2 + 3); and level three nodes store prefixes whose length is 7 (2 + 3 + 2). This poses a problem for the prefixes of our example, because the length of some of these prefixes is different from the storeable lengths.



(a) 8-prefix example of [30]

(b) Corresponding 1-bit trie

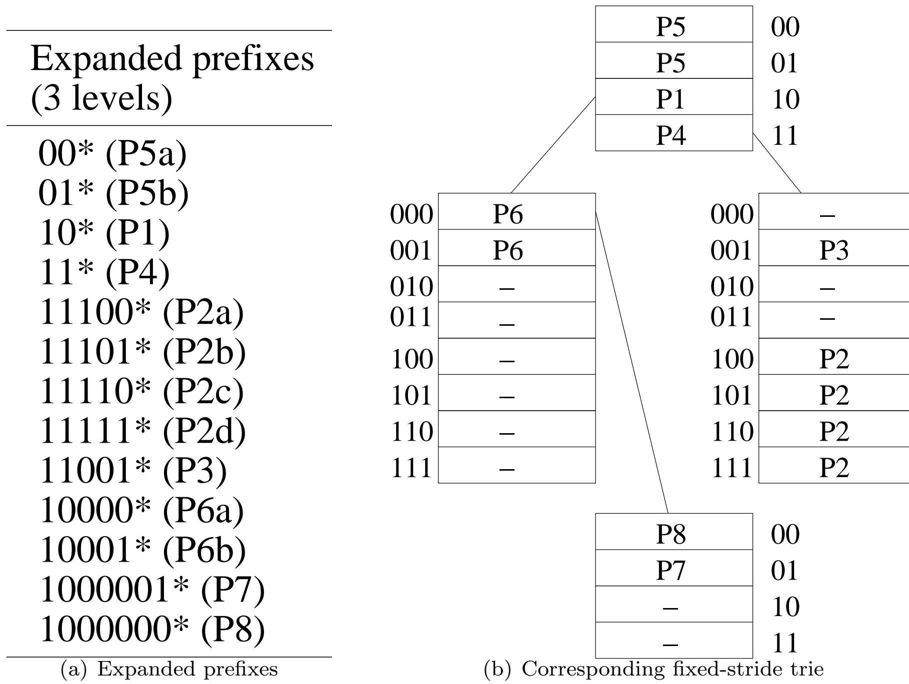
Figure 1: Prefixes and corresponding 1-bit trie [25]

For instance, the length of P5 is 1. To get around this problem, a prefix with a nonpermissible length is expanded to the next permissible length [30]. For example, P5 = 0* is expanded to P5a = 00* and P5b = 01*. If one of the newly created prefixes is a duplicate, dominance rules are used to eliminate all but one occurrence of the prefix. Because of the elimination of duplicate prefixes from the expanded prefix set, all prefixes are distinct. Figure 2(a) shows the prefixes that result when we expand the prefixes of Figure 1 to lengths 2, 5, and 7. Duplicate prefixes, following expansion, are eliminated by favoring longer length original prefixes when longest length prefix matching is desired. Figure 2(b) shows the corresponding FST whose height is 2 and whose strides are 2, 3, and 2.

Since the trie of Figure 2(b) can be searched with at most 3 memory accesses, it represents a time-performance improvement over the 1-bit trie of Figure 1(b), which requires up to 7 memory accesses to perform a search. However, the space requirements of the FST of Figure 2(b) are more than that of the corresponding 1-bit trie. For the root of the FST, we need 4 units; the two level 1 nodes require 8 units each; and the level 3 node requires 4 units. The total is 24 memory units. Note that the 1-bit trie of Figure 1 requires only 20 memory units.

Let N be a node at level j of a multibit trie. Let s_0, s_1, $\cdots$, s_j be the strides of the nodes on the path from the root of the multibit trie to node N. Note that s_0 is the stride of the root and s_j is the stride of N. With node N we associate a pair $[s, e]$, called the *start-end* pair, that gives the start and end levels of the corresponding 1-bit trie O covered by this node. By definition, $s = \sum_{i=0}^{j-1} s_j$ and $e = s + s_j - 1$. The root of the multibit trie *covers* levels 0 through $s_0 - 1$ of O and node N covers levels 0 through $s_j - 1$ of a corresponding subtrie of O. In the case of an FST all nodes at the same level of the FST have the same $[s, e]$ values. In the FST of Figure 2(b), the $[s, e]$ values for the level 0, level 1, and level 2 nodes are $[0, 1]$, $[2, 4]$ and $[5, 6]$, respectively. Level 0 of the FST covers levels 0 and 1 of the corresponding 1-bit trie while level 2 of this FST covers levels 2, 3, and 4 of the 1-bit trie. Levels 0 and 1 of the FST together cover levels 0 through 4 of the 1-bit trie.

Starting with a 1-bit trie for n prefixes whose length is at most W, the strides for a space-optimal FST with



Expanded prefixes (3 levels)
00* (P5a)
01* (P5b)
10* (P1)
11* (P4)
11100* (P2a)
11101* (P2b)
11110* (P2c)
11111* (P2d)
11001* (P3)
10000* (P6a)
10001* (P6b)
1000001* (P7)
1000000* (P8)

(a) Expanded prefixes

(b) Corresponding fixed-stride trie

Figure 2: Prefix expansion and fixed-stride trie

at most k levels may be determined in $O(nW + kW^2)$ time[1] [30, 25]. For a space-optimal VST whose height[2] is constrained to k, the strides may be determined in $O(nW^2k)$ time [30, 25].

2.3 Two-Dimensional 1-Bit Tries

A *two-dimensional 1-bit trie* (2D1BT) is a one-dimensional 1-bit trie (called the top-level trie) in which the data field of each element[3] is a pointer to a (possibly empty) 1-bit trie (called the lower-level trie). So, a 2D1BT has 1 top-level trie and potentially many lower-level tries.

Consider the 7-rule two-dimensional classifier of Figure 3. For each rule, the filter is defined by the Dest (destination) and Source prefixes. So, for example, $F1 = (0*, 1100*)$ matches all packets whose destination address begins with 0 and whose source address begins with 1100. When a packet is matched by two or more filters, the rule with least cost is used. The classifier of Figure 3 may be represented as a 2D1BT in which the top-level trie is constructed using the destination prefixes. In the context of our destination-source filters, this top-level trie is called the *destination trie* (or simply, dest trie). Let N be a node in the destination trie and let E be one of the element fields (either le or re) of N. If no dest prefix equals $Q(E)$, then $N.E.data$ points to an empty lower-level trie. If there is a dest prefix D that equals $Q(E)$, then $N.E.data$ points to a 1-bit trie for all source prefixes S such

[1]The complexity of $O(kW^2)$ given in [30, 25] assumes we start with data extracted from the 1-bit trie; the extraction of this data takes $O(nW)$ time.

[2]The height of a trie is the number of levels in the trie. Height is often defined to be 1 less than the number of levels. However, the definition we use is more convenient in this paper.

[3]Recall that each node of a 1-bit trie has two element fields.

Filter	Dest	Source	Cost	Action
F1	0*	1100*	1	Allow inbound mail
F2	0*	1110*	2	Allow DNS access
F3	0*	1111*	3	Secondary access
F4	000*	10*	4	Incoming telnet
F5	000*	11*	5	Outgoing packets
F6	0001*	000*	6	Return ACKs okay
F7	0*	1*	7	Block packet

Figure 3: An example of seven dest-source filters

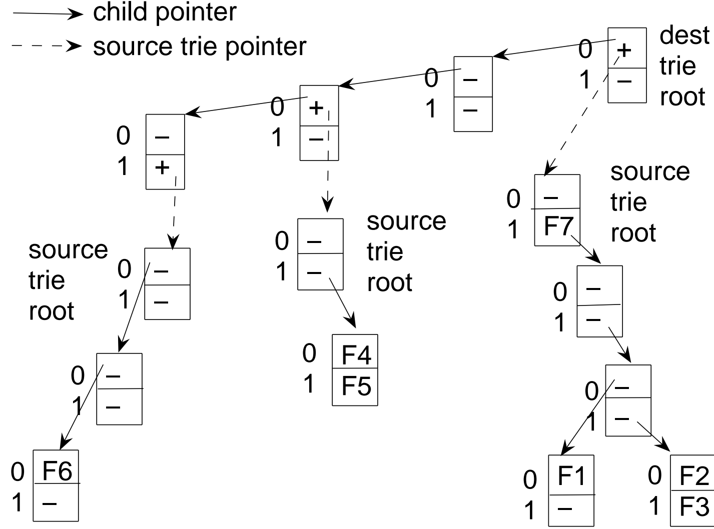


Figure 4: Two-dimensional 1-bit trie for Figure 3

that (D, S) is a filter. In the context of destination-source filters, the lower-level tries are called *source tries*. We say that N has two source-tries (one or both of which may be empty) *hanging* from it. Figure 4 gives the 2D1BT for the filters of Figure 3.

2.4 Two-Dimensional Multibit Tries

Two-dimensional multibit tries (2DMTs) [19] are a natural extension of 2D1BTs to the case when nodes of the dest and source tries may have a stride that is more than 1. As in the case of one-dimensional multibit tries, prefix expansion is used to accomodate prefixes whose length lies between the $[s, e]$ values of a node on its search path. Let $D_1, D_2, \cdots, D_u$ be the distinct destination prefixes in the rule table. For the rules of Figure 3, $u = 3$, $D_1 = 0*$, $D_2 = 000*$ and $D_3 = 0001*$. The destination trie of the 2DMT for our 7 filters is a multibit trie for D_1–D_3 (see Figure 5). In the destination trie of this figure, element fields that correspond to a D_i or the expansion of a D_i (in case D_i is to be expanded) are marked with '+'. The remaining element fields are marked with '−'. Element fields marked with '+' have a non-empty source trie hanging from them, the remaining element fields have an empty (or

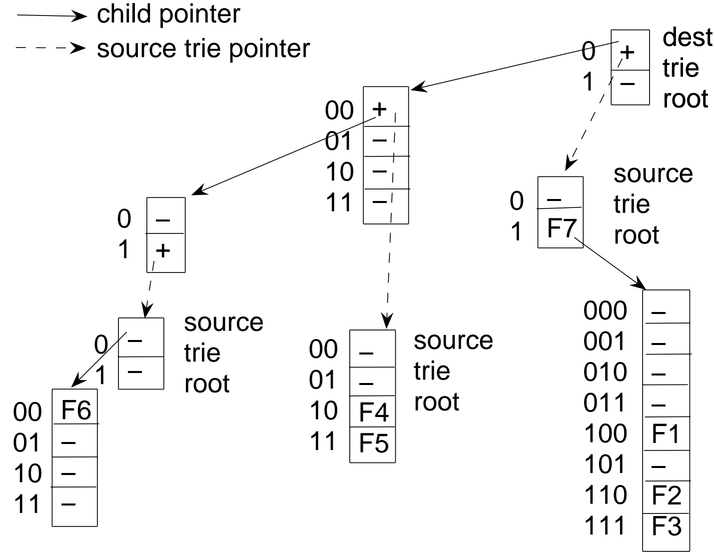


Figure 5: 2DMT for filters of Figure 3

null) source trie hanging from them.

Let $L(p)$ be the length of the prefix p. To define the source tries of a 2DMT, we introduce the following terminology:

- $D_{i,j} = \{D_q | D_q$ is a prefix of D_i and $L(D_i) - L(D_q) \leq j\}$
- $S_{i,j} = \{S_q | (D_q, S_q)$ is a filter and $D_q \in D_{i,j}\}$

We see that $D_{i,0} = \{D_i\}$ for all i. For our example filter set, $D_{1,0} = \{0*\}$, $S_{1,0} = \{1*, 1100*, 1110*, 1111*\}$, $D_{2,0} = D_{2,1} = \{000*\}$, $S_{2,0} = S_{2,1} = \{10*, 11*\}$, $D_{2,2} = \{0*, 000*\}$, $S_{2,2} = \{1*, 10*, 11*, 1100*, 1110*, 1111*\}$, $D_{3,0} = \{0001*\}$, $S_{3,0} = \{000*\}$, $D_{3,1} = D_{3,2} = \{0001*, 000*\}$, $S_{3,1} = S_{3,2} = \{000*, 10*, 11*\}$, $D_{3,3} = \{0001*, 000*, 0*\}$, and $S_{3,3} = \{000*, 10*, 11*, 1100*, 1110*, 1111*\}$.

Let N be a node in the destination trie of a 2DMT. Let $[s, e]$ be the start-end pair associated with N and let E be an element field of N. Let D_i be the longest destnation prefix, $s < L(D_i) \leq e + 1$, that expands (note that a prefix may expand to itself, a trivial expansion) to $Q(E)$. If there is no such D_i, the source trie that hangs from E is empty. If D_i exists, the source trie that hangs from E is a multibit trie for $S_{i,j}$, where $j = L(D_i) - s - 1$. When multiple filters expand to the same element field of the multibit source trie, the least-cost filter is recorded. For our example of Figure 5, the source tries hanging from the '+' fields of the three dest-trie nodes (top to bottom) are $S_{1,0}$, $S_{2,1}$ and $S_{3,0}$. *Note that, in a dest-trie node of a 2DMT, several Es may have the same source trie hanging from them. To conserve space, only one copy of each distinct source trie is retained.*

A 2DMT may be searched for the least-cost filter that matches a given pair of destination and source addresses (da, sa) by following the search path for da in the destination trie of the 2DMT. All source tries encountered on this path are searched for sa. The least-cost filter recorded on these source-trie search paths is the least-cost filter

that matches (da, sa). Suppose we are to find the least-cost filter that matches $(00010, 11101)$. Searching the 2DMT of Figure 5 for 00010 takes us through the 0 field of the root, the 00 field of the root's left child, and the 1 field of the remaining dest-trie node. Each of these three fields has a non-empty source-trie hanging from it. So, three source tries are searched for 11101. When the root source-trie is searched, the element fields with $F2$ and $F7$ are encountered. When the source trie in the left child of the root is searched, the element field with $F5$ is encountered. The search in the remaining source trie encounters no matching filters. The least-cost matching-filter is determined to be $F2$.

3 Pipelined One-Dimensional VSTs

3.1 A Dynamic Programming Construction

Algorithms to construct optimal pipelined one-dimensional FSTs (i.e., FSTs that satisfy constraints C1–C3) have been developed in [16]. So, we consider only the construction of pipelined one-dimensional VSTs in this section. Optimal pipelined VSTs satisfy constraints C1–C3 (although, in C1, we replace "fixed-stride trie" with "variable-stride trie"). Although we do not develop an algorithm that constructs an optimal pipelined VST, the heuristic proposed by us, in this section, constructs an "approximately optimal" VST, which when mapped onto a pipelined architecture using constraint C1 results in a maximum per-stage memory requirement that is considerably less than that for an optimal pipelined FST for the given rule table.

Let O be the 1-bit trie for the given filter set, let N be a node of O and let $ST(N)$ be the subtree of O that is rooted at N. Let $Opt'(N, r)$ denote the approximately optimal (pipelined) VST for the subtree $ST(N)$; this VST has at most r levels. Let $Opt'(N, r).E(l)$ be the (total) number of elements at level l of $Opt'(N, r)$, $0 \le l < r$. We seek to construct $Opt'(root(O), k)$, the approximately optimal pipelined VST for O that has at most k levels, where k is the number of available pipeline stages.

Let $D_i(N)$ denote the descendents of N that are at level i of $ST(N)$. So, for example, $D_0(N) = \{N\}$ and $D_1(N)$ denotes the children, in O, of N. Our approximately optimal VST has the property that its subtrees are approximately optimal for the subtrees of N that they represent. So, for example, if the root of $Opt'(N, r)$ represents levels 0 through $i - 1$ of $ST(N)$, then the subtrees of (the root of) $Opt'(N, r)$ are $Opt'(M, r - 1)$ for $M \in D_i(N)$.

When $r = 1$, $Opt'(N, 1)$ has only a root node; this root represents all levels of $ST(N)$. So,

$$Opt'(N, 1).E(l) = \begin{cases} 2^{height(N)} & l = 0 \\ 0 & l > 0 \end{cases} \tag{1}$$

where $height(N)$ is the height of $ST(N)$.

When $r > 1$, the number, q, of levels of $ST(N)$ represented by the root of $Opt'(N, r)$ is between 1 and $height(N)$. From the definition of $Opt'(N, r)$, it follows that

$$Opt'(N, r).E(l) = \begin{cases} 2^q & l = 0 \\ \sum_{M \in D_q(N)} Opt'(M, r - 1).E(l - 1) & 1 \le l < r \end{cases} \tag{2}$$

where q is as defined below

$$q = argmin_{1 \leq i \leq height(N)}\{\max\{2^i, \max_{0 \leq l < r-1}\{\sum_{M \in D_i(N)} Opt'(M, r-1).E(l)\}\}\} \tag{3}$$

Although the dynamic programming recurrences of Equations 1–3 may be solved directly to determine $Opt'(root(O), k)$, the time complexity of the resulting algorithm is reduced by defining auxilliary equations. For this purpose, let $Opt'STs(N, i, r-1)$, $i > 0$, $r > 1$, denote the set of approximately optimal VSTs for $D_i(N)$ ($Opt'STs(N, i, r-1)$ has one VST for each member of $D_i(N)$); each VST has at most $r-1$ levels. Let $Opt'STs(N, i, r-1).E(l)$ be the sum of the number of elements at level l of each VST of $Opt'STs(N, i, r-1)$. So,

$$Opt'STs(N, i, r-1).E(l) = \sum_{M \in D_i(N)} Opt'(M, r-1).E(l), 0 \leq l < r-1 \tag{4}$$

For $Opt'STs(N, i, r-1).E(l)$, $i > 0$, $r > 1$, $0 \leq l < r-1$, we obtain the following recurrence

$$Opt'STs(N, i, r-1).E(l) = \begin{cases} Opt'(LC(N), r-1).E(l) + Opt'(RC(N), r-1).E(l) & i = 1 \\ Opt'STs(LC(N), i-1, r-1).E(l) + Opt'STs(RC(N), i-1, r-1).E(l) & i > 1 \end{cases} \tag{5}$$

where $LC(N)$ and $RC(N)$, respectively, are the left and right children, in $ST(N)$, of N.

Since the number of nodes in O is $O(nW)$, the total number of $Opt'(N, r)$ and $Opt'STs(N, i, r-1)$ values is $O(nW^2k)$. For each, $O(k)$ $E(l)$ values are computed. Hence, to compute all $Opt'(root(O), k).E(l)$ values, we must compute $O(nW^2k^2)$ $Opt'(N, r).E(l)$ and $Opt'STs(N, i, r-1).E(l)$ values. Using Equations 1–5, the total time for this is $O(nW^2k^2)$.

3.2 Mapping Onto A Pipeline Architecture

When the approximately optimal VST $Opt'(root(O), k)$ of Section 3.1 is mapped onto a k stage pipeline in the most straightforward way (i.e., nodes at level l of the VST are packed into stage $l+1$, $0 \leq l < k$ of the pipeline), the maximum per-stage memory is

$$\max_{0 \leq l < k}\{Opt'(root(O), k).E(l)\}$$

We can do quite a bit better than this by employing a more sophisticated mapping strategy. For correct pipeline operation, we need require only that if a node N of the VST is assigned to stage q of the pipeline, then each descendent of N be assigned to a stage r such that $r > q$. Hence, we are motivated to solve the following tree packing problem:

Tree Packing (TP)

Input: Two integers $k > 0$ and $M > 0$ and a tree T, each of whose nodes has a positive size.

Output: "Yes" iff the nodes of T can be packed into k bins, each of capacity M. The bins are indexed 1 through k and the packing is constrained so that for every node packed into bin q, each of its descendent nodes is packed

into a bin with index more than q.

By performing a binary (or other) search over M, we may use an algorithm for TP to determine an optimal packing (i.e., one with least M) of $Opt'(root(O), k)$ into a k-stage pipeline. Unfortunately, problem TP is NP-complete. This may be shown by using a reduction from the partition problem [13]. In the partition problem, we are given n positive integers s_i, $1 \leq i \leq n$ whose sum is $2B$ and we are to detemine whether any subset of the given s_is sums to B.

Theorem 1 *TP is NP-complete.*

Proof It is easy to see that TP is in NP. So we simply show the reduction from the partition problem. Let n, s_i, $1 \leq i \leq n$, and B ($\sum s_i = 2B$) be an instance of the partition problem. We may transform, in polynomial time, this partition instance into a TP instance that has a k-bin tree packing with bin capacity M iff there is a partition of the s_is. The TP instance has $M = 2B + 1$ and $k = 3$. The tree T for this instance has three levels. The size of the root is M; the root has n children; the size of the ith child is $2s_i$, $1 \leq i \leq n$; and the root has one grandchild whose size is 1 (the grandchild may be be made a child of any one of the n children of the root).

It is easy to see that T may be packed into 3 capacity M bins iff the given s_is have a subset whose sum is B. ■

All VSTs have nodes whose size is a power of 2 (more precisely, some constant times a power of 2). The TP construction of Theorem 1 results in node sizes that are not necessarily a power of 2. Despite Theorem 1, it is possible that TP restricted to nodes whose size is a power of 2 is polynomially solvable. However, we have been unable to develop a polynomial-time algorithm for this restricted version of TP. Instead, we propose a heuristic, which is motivated by the optimality of the First Fit Decreasing (FFD) algorithm to pack bins when the size of each item is a power of a, where $a \geq 2$ is an integer. In FFD [13], items are packed in decreasing order of size; when an item is considered for packing, it is packed into the first bin into which it fits; if the item fits in no exisiting bin, a new bin is started. Although this packing strategy does not guarantee to minimize the number of bins into which the items are packed when item sizes are arbitrary integers, the strategy works for the restricted case when the size of each item is of the form a^i, where $a \geq 2$ is an integer. Theorem 2 establishes this by considering a related problem–restricted max packing (RMP). Let $a \geq 2$ be an integer. Let s_i, a power of a, be the size of the ith item, $1 \leq i \leq n$. Let c_i be the capacity of the ith bin, $1 \leq i \leq k$. In the *restricted max packing* problem, we are to maximize the sum of the sizes of the items packed into the k bins. We call this version of max packing restricted because the item sizes must be a power of a.

Theorem 2 *FFD solves the RMP problem.*

Proof Let a, n, c_i, $1 \leq i \leq k$ and s_i, $1 \leq i \leq n$ define an instance of RMP. Suppose that in the FFD packing the sum of the sizes of items packed into bin i is b_i. Clearly, $b_i \leq c_i$, $1 \leq i \leq k$. Let S be the subset of items not packed in any bin. If $S = \emptyset$, all items have been packed and the packing is necessarily optimal. So, assume that

$S \neq \emptyset$. Let A be the size of smallest item in S. Let x_i and y_i be non-negative integers such that $b_i = x_i A + y_i$ and $0 \leq y_i < A$, $1 \leq i \leq k$. We make the following observations:

(a) $(x_i + 1) * A > c_i$, $1 \leq i \leq k$. This follows from the definition of FFD and the fact that S has an unpacked item whose size is A.

(b) Each item that contributes to a y_i has size less than A. This follows from the fact that all item sizes (and hence A) are a power of a. In particular, note that every item size $\geq A$ is a multiple of A.

(c) Each item that contributes to the $x_i A$ component of a b_i has size $\geq A$. Though at first glance, it may seem that many small items could collectively contribute to this component of b_i, this is not the case when FFD is used on items whose size is a power of a. We prove this by contradiction. Suppose that for some i, $x_i A = B + C$, where $B > 0$ is the contribution of items whose size is less than A and C is the contribution of items whose size is $\geq A$. As noted in (b), every size $\geq A$ is a multiple of A. So, C is a multiple of A. Hence B is a multiple of A formed by items whose size is smaller than A. However S has an item whose size is A. FFD should have packed this item of size A into bin i before attempting to pack the smaller size items that constitute B.

The sum of the sizes of items packed by FFD is

$$FFDSize = \sum_{1 \leq i \leq k} x_i A + \sum_{1 \leq i \leq k} y_i \tag{6}$$

For any other k-bin packing of the items, let $b'_i = x'_i A + y'_i$, where x'_i and y'_i are non-negative integers and $0 \leq y'_i < A$, $1 \leq i \leq k$, be the sum of the sizes of items packed into bin i. For this other packing, we have

$$OtherSize = \sum_{1 \leq i \leq k} x'_i A + \sum_{1 \leq i \leq k} y'_i \tag{7}$$

From observation (a), it follows that $x'_i \leq x_i$, $1 \leq i \leq k$. So, the first sum of Equation 7 is $\leq$ the first sum of Equation 6.

From observations (b) and (c) and the fact that A is the smallest size in S, it follows that every item whose size is less than A is packed into a bin by FFD and contributes to the second sum in Equation 6. Since all item sizes are a power of a, no item whose size is more than A can contribute to a y'_i. Hence, the second sum of Equation 7 is $\leq$ the second sum of Equation 6. So, $OtherSize \leq FFDSize$ and FFD solve the RMP problem. ■

The optimality of FFD for RMP motivates our tree packing heuristic of Figure 6, which attempts to pack a tree into k bins each of size M. It is assumed that the tree height is $\leq k$. The heuristic uses the notions of a ready node and a critical node. A *ready node* is one whose ancestors have been packed into prior bins. Only a ready node may be packed into the current bin. A *critical node* is an, as yet, unpacked node whose height[4] equals the number of bins remaining for packing. Clearly, all critical nodes must be ready nodes and must be packed into the current bin if we are to successfully pack all tree nodes into the given k bins. So, our heuristic ensures that critical

[4]The height of a leaf is 1; the height of a non-leaf is 1 more than the maximum of the heights of its children.

Step 1: [Initialize]
$currentBin = 1$; $readyNodes =$ tree root;

Step 2: [Pack into current bin]
Pack all critical ready nodes into $currentBin$;
if bin capacity is exceeded **return failure**;
Pack remaining ready nodes into $currentBin$ in decreasing order of node size;

Step 3 [Update Lists]
if all tree nodes have been packed **return** success;
if $currentBin == k$ **return failure**;
Remove all nodes packed in Step 2 from $readyNodes$;
Add to $readyNodes$ the children of all nodes packed in Step 2;
$currentBin++$;
Go to Step 2;

Figure 6: Tree packing heuristic

nodes are ready nodes. Further, it first packs all critical nodes into the current bin and then packs the remaining ready nodes in decreasing order of node size. We may use the binary search technique to determine the smallest M for which the heuristic is successful in packing the given tree.

4 Pipelined 2DMTs–Coarse Mapping

We consider two different strategies to map a 2DMT onto a pipelined architecture–coarse- and fine-grain. In a coarse-grain mapping each node of the dest trie together with the source tries that hang from it are considered as an indivisible unit and assigned to a single pipeline stage. Using a coarse-grain mapping, the 2DMT of Figure 5 has a unique mapping onto a 3 stage pipeline, each level of the dest trie is mapped to a different stage of the pipeline. In this mapping, the dest-trie root and the 2-node source subtrie that hangs from it map into stage 1 of the pipeline, the left child of the dest-trie root together with its 1-node hanging source trie map into stage 2, the level 2 dest-trie node and it 2-node hanging source trie map into stage 3. The memory requirement for the 3 stages is 12 (2 for the dest-trie node and 2 + 8 for the 2 source-trie nodes), 8, and 8, respectively. The maximum per-stage memory, therefore, is 12. The maximum number of memory accesses needed to process a packet is 3 for stage 1 (1 to access the dest-trie node and 1 for each of the 2 source-trie nodes), 2 for stage 2, and 3 for stage 3. For smooth operation of the pipeline, the cycle time has to be sufficent for 3 memory accesses.

In a fine-grain mapping, a dest-trie node and the nodes of the source tries hanging from it are mapped to different stages of the pipeline. So, for example, in a fine-grain mapping of the 2DMT of Figure 5 onto an 8-stage pipline, the dest-trie root could be mapped to stage 1, the root of its hanging source trie to stage 2 and the remaining node in this source trie to stage 3; the level 1 dest-trie node could be mapped to stage 4 and its 1-node hanging source trie to stage 5; the remaining 3 stages of the pipeline could be used for the level 2 dest-trie node and its 2-node hanging source trie. Each stage would perform one memory access when processing a packet and

the maximum per-stage memory is 8.

In this section, we consider coarse-grain mapping only. Fine-grain mapping is considered in Section 5. we develop two algorithms to construct 2DMTs that are suitable for a coarse-grain mapping onto a k-stage pipeline architecture. The first constructs a 2DMT in which the dest trie is an FST and the source tries are VSTs. The constructed 2DMT is optimal (in the sense of constraints C1–C3) for coarse-grain mapping under the assumption that the 2DMT dest trie is an FST. In the second algorithm, the constructed 2DMT is a VST. When the constructed 2DMT is mapped so as to satisfy C1 (under the added constraint of a coarse-grain mapping), constraints C2 and C3 may not be satisfied. However, experimental results reported in Section 6 indicate that the mapping requires much less maximum per-stage as well as total memory than when the 2DMT of the first algorithm is used.

4.1 FST Dest Trie

Assume that the dest trie of the pipelined 2DMT is an FST while the source tries are VSTs. Further, assume that each pipeline stage has a memory access budget of $H + 1$. With this budget, each VST source trie that hangs off of a dest-trie node has a height of at most H (the additional memory access being allocated to the access of the dest-trie node). Let O be the 2D1BT for the filter set. Let $T(j, r)$ be an r-level 2DMT (i.e., the dest trie of T is an r-level FST) that covers levels 0 through j of O; the source tries that hang off of the dest-trie nodes of T are space-optimal VSTs that have at most H levels each. Let (s, e) be the start-end pair associated with some level l of the dest-trie FST of T. Let $sourceSum(s, e)$ be the total number of elements in the space-optimal VSTs that hang off of all of the level l dest-trie nodes (note that these VSTs have at most H levels each). The number of elements, $E(T(j, r), l)$ on level l of T is defined to be the number of elements in the dest-trie nodes at level l plus the number of elements in all the source tries that hang off of these level l dest-trie nodes. So,

$$E(T(j, r), l) = nodes(s) * 2^{e-s+1} + sourceSum(s, e)$$

where $nodes(s)$ is the number of nodes at level s of O. Let $ME(T(j, r)) = \max_l\{E(T(j, r), l)\}$ be the maximum number of elements on any level of T and let $MME(j, r)$ be the minimum value for $ME(T(j, r))$ taken over all possible 2DMTs $T(j, r)$. Note that $MME(W - 1, k)$ is the minimum per-stage memory required by a coarse pipeline mapping of the optimal pipelined 2DMT for O (this 2DMT is constrained so that the dest trie is a k level FST and each source-trie is a VST whose height is at most H).

We obtain a dynamic programming recurrence for MME. First, note that when $r = 1$, levels 0 through j of O must be represented by a single level of the 2DMT; this level has a single node, the root, whose stride is $j + 1$ (its, (s, e) pair is $(0, j)$). When, $r > 1$, the last level of the dest trie covers levels $m + 1$ through j of O for some m in the range $[r - 2, j - 1]$ and the remaining $r - 1$ levels of the dest trie cover levels 0 through m of O. Using these observations and the definition of MME, we obtain

$$MME(j, r) = \begin{cases} 2^{j+1} + sourceSum(0, j) & r = 1 \\ \min_{r-2 \le m \le j-1} \max\{MME(m, r-1), nodes(m+1) * 2^{j-m} + sourceSum(m+1, j)\} & r > 1 \end{cases} \quad (8)$$

To determine $MME(W-1,k)$, we first compute $O(W^2)$ $sourceSum$ values. This can be done in $O(n^2W^2)$ time using the algorithm *sourceTries* of [19]. Then, $O(kW)$ MME values are computed using Equation 8. Each of these MME values is computed in $O(W)$ time, for a total of $O(kW^2)$ time. The overall time to compute $MME(W-1,k)$ is, therefore, $O(n^2W^2+kW^2)=O(n^2W^2)$ (under the very realistic assumption that $k=O(n^2)$).

Once we have determined $MME(W-1,k)$, the 2DMT with minimum total number of elements subject to the constraint that its MME value is $MME(W-1,k)$ may be determined using another dynamic programming recurrence. Let $TE(j,r)$ be the minimum number of elements in any r-level 2DMT that covers levels 0 through j of O; the 2DMT is constrained so that the optimal VSTs that hang off of each dest-trie node have at most H levels and the MME value of the 2DMT is at most $MME(W-1,k)$. It is easy to see that

$$TE(j,1)=\begin{cases} 2^{j+1}+sourceSum(0,j) & \text{if } 2^{j+1}+sourceSum(0,j)\leq MME(W-1,k)\\ \infty & \text{otherwise}\end{cases} \tag{9}$$

For $r>1$, we get

$$TE(j,r)=\min_{m\in X(j,r)} \max\{TE(m,r-1)+nodes(m+1)*2^{j-m}+sourceSum(m+1,j)\}\} \tag{10}$$

where $X(j,r)=\{m|r-2\leq m\leq j-1 \text{ and } nodes(m+1)*2^{j-m}+sourceSum(m+1,j)\leq MME(W-1,k)\}$.

The additional time needed to compute $TE(W-1,k)$ using Equations 9 and 10 is $O(kW^2)$. Note that the tree-mapping heuristic of Figure 6 may be emploed to map the constructed trie on to a k stage pipeline.

4.2 VST Dest Trie

We extend our heuristic of Section 3 to obtain approximately optimal coarse-grain pipelined 2DMTs whose dest and source tries are VSTs; each source VST has at most H levels. Let O be the 2D1BT for the given filter set, let N be a node of the dest trie of O and let $ST(N)$ be the subtree of O that is rooted at N. Note that $ST(N)$ includes all dest-trie nodes of O that are descendents of N together with the source tries that hang off of these dest-trie nodes. Let $Opt'(N,r)$ denote the approximately optimal (pipelined) 2DMT for the subtree $ST(N)$; the dest trie of this 2DMT is a VST that has at most r levels, each of the source VSTs hanging off of the dest-trie nodes has at most H levels. Let $Opt'(N,r).E(l)$ be the (total) number of elements at level l of $Opt'(N,r)$, $0\leq l<r$ (this includes elements of source tries that hang off of level l dest-trie nodes). We seek to construct $Opt'(root(O),k)$.

Let $D_i(N)$ denote the descendents of N that are at level i of the dest trie of $ST(N)$. As in Section 3, our approximately optimal 2DMT has the property that the dest-trie subtrees are approximately optimal for the subtrees of N that they represent.

When $r=1$, $OPT'(N,1)$ has only a root node; this root represents all of $ST(N)$. So,

$$Opt'(N,1).E(l)=\begin{cases} 2^{h(N)}+sourceSum(N,h(N)-1) & l=0\\ 0 & l>0\end{cases} \tag{11}$$

where $h(N)$ is the height of the dest trie of $ST(N)$ and $sourceSum(N,j)$ is the total number of elements in the space-optimal VSTs that hang off of a dest-trie node that covers levels 0 through j of $ST(N)$; each VST has at most H levels.

When $r > 1$, the number of levels of the dest trie of $ST(N)$ represented by the root of $Opt'(N, r)$ is between 1 and $h(N)$. From the definition of $Opt'(N, r)$, it follows that

$$Opt'(N,r).E(l) = \begin{cases} 2^q + sourceSum(N, q-1) & l = 0 \\ \sum_{M \in D_q(N)} Opt'(M, r-1).E(l-1) & 1 \leq l < r \end{cases} \quad (12)$$

where q is as defined below

$$q = argmin_{1 \leq i \leq h(N)} \{\max\{2^i + sourceSum(N, i-1), \max_{0 \leq l < r-1} \{\sum_{M \in D_i(N)} Opt'(M, r-1).E(l)\}\}\} \quad (13)$$

As was the case in Section 3, the time needed to determine $Opt'(root(O), k)$ is reduced by defining auxilliary equations. For this purpose, let $Opt'STs(N, i, r-1)$, $i > 0$, $r > 1$, denote the set of approximately optimal pipelined 2DMTs for $D_i(N)$ ($Opt'STs(N, i, r-1)$ has one 2DMT for each member of $D_i(N)$), the dest trie of each 2DMT has at most $r - 1$ levels and each source trie has at most H levels. Let $Opt'STs(N, i, r-1).E(l)$ be the sum of the number of elements at level l of each 2DMT of $Opt'STs(N, i, r-1)$. Equations 4 and 5 hold for the new definition of $Opt'STs$.

All values of $sourceSum(N, j)$ may be computed in $O(n^2W^3k)$ time as in [19]. Using Equations 11–13, 4 and 5, we may compute $Opt'(root(O), k)$ in an additional $O(nW^2k^2)$ time. So, the total time needed to determine the approximately optimal pipelined 2DMT in which the dest and source tries are VSTs is $O(n^2W^3k + nW^2k^2) = O(n^2W^3k)$ (note that $k \leq W$ for a coarse grain mapping).

Enhanced Computation of $Opt'STs(N, i, j)$

We can improve the pipelined 2DMT tries constructed using Equations 11–13, 4 and 5 by employing the node pullup technique [4, 16]. The node pullup techique helps reduce congestion (i.e., excess memory required by a trie level) at a trie level by collapsing subtries at that level into the parent level.

As an example, consider the 2D1BT $ST(R)$ of Figure 7. Suppose that the optimal H-level VST for each of the source tries $S1$ and $S2$ has 100 elements. When Equations 11–13, 4 and 5 are used, the constructed 3-stage pipelined 2DMT for $ST(R)$ has node R assigned to stage 1, nodes A and B assigned to stage 2, and nodes C and D together with the source tries $S1$ and $S2$ assigned to stage 3. The memory required by the 3 stages is 2 (the stride of R is 1), 4, and 204 (2 for each of C and D and 100 for each of $S1$ and $S2$), respectively. Using a node pullup, we can pull D into B (or C into A) changing the stride of the right (left) child of R to 2. When the resulting 2DMT is mapped onto a 3-stage pipeline, the memory requirement for stage 1 is 2, that for stage 2 is 2 (for the single stride 1 dest node at level 1) + 4 (for the stride 2 dest node at level 1) + 100 (for the single distinct non-empty source trie hanging from the stride 2 dest node) = 106, and 2 (for the single dest-trie node at level 2 of the dest trie) + 100 (for the source trie hanging from this dest-trie node) = 102. For $ST(R)$, the performed node pullup reduces the maximum per-stage memory from 204 to 106.

Let N be a dest-trie node of O. When computing $Opt'STs(N, i, j)$, for each $M \in D_i(N)$, we may either do a node pullup (i.e., $ST(M)$ is represented by a single dest-trie node in the 2DMT) or represent $ST(M)$ by its

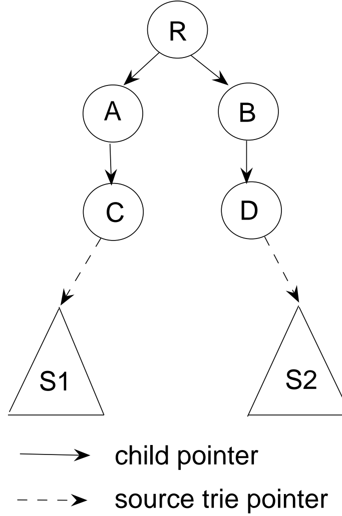


Figure 7: Example for node pullup

approximately optimal j-level 2DMT whose maximum per-stage memory requirement is $Opt'(M, j)$. Algorithm $NodePullup$ (Figure 8) uses a greedy strategy to choose between these two options. The algorithm computes $STs(N, i, j).E(j)$ using $Opt'(M, j)$ for $M \in D_i(N)$. Note that the algorithm incorporates the node pullup option directly into Equation 4.

5 Pipelined 2DMTs–Fine Mapping

As in Section 4, we consider two cases. In the first, the dest trie of the 2DMT used to map onto the pipeline architecture is an FST and in the second, this dest trie is a VST. Unlike in Section 4 where we obtained an algorithm to construct an optimal pipelined 2DMT (under the constraint that the dest trie is an FST), our algorithms for fine grain mapping do not guarantee optimality. Hence, they are just heuristics to construct good pipelined 2DMTs for fine grain mapping.

5.1 FST Dest Trie

Let (s, e) denote the start-end pair associated with some level of an FST dest trie for O. Let $S(s, e)$ be the set of distinct 1-bit source tries that hang off of this level of the FST and let $T(s, e, h)$ be a set of multibit tries (either FST or VST) for $S(s, e)$; the height of each multibit trie is at most h. Note that $T(s, e, h)$ has one multibit trie for each 1-bit trie in $S(s, e)$. For any multibit trie U, let $U.E(l)$ be the total number of elements on level l of U and let $U.size$ be the total number of elements in U. Let

$$maxE(T(s, e, h)) = \max_{l}\{\sum_{T_i \in T(s,e,h)} T_i.E(l)\}$$

```
Algorithm NodePullup(N, i, j)
// N is a node in the dest trie of O.
// i, i > 0, represents a level of ST(N).
// j > 0 is the number of levels in the 2DMT for D_i(N).
// Return Opt'STs(N, i, j).E(l), 0 <= l < j.
{
    for (v = 0; v < j; v++) Opt'STs(N, i, j).E(v) = 0;
    curMaxE = 0; // curMaxE represents max_{0<=l<j}{Opt'STs(N, i, j).E(l)}.
    for (each M ∈ D_i(N)) do {
        choice1 = max{(Opt'(M, 1).E(0) + Opt'STs(N, i, j).E(0), curMaxE};
            // node pullup
        for (v = 0; v < j; v++) tmp(v) = Opt'STs(N, i, j).E(v) + Opt'(M, j).E(v);
        choice2 = max_{0<=l<j}{tmp(l)};
        if (choice1 <= choice2){
            // Do a node pullup.
            Opt'STs(N, i, j).E(0)+ = Opt'(M, 1).E(0);
            curMaxE = choice1;
        }
        else {
            // Use Opt'(M, j) for ST(M).
            for (v = 0; v < j; v++) Opt'STs(N, i, j).E(v) = tmp(v);
            curMaxE = choice2;
        }
    }
}
```

Figure 8: Computing $Opt'STs(N, i, j)$ using node pullups

Define $minMaxE$ and $totalE$ as below:

$$minMaxE(s, e, h) = \min\{maxE(T(s, e, h))|T(s, e, h) \text{ is for } S(s, e)\}$$

$$totalE(s, e, h, p) = \min\{\sum_{T_i \in T(s,e,h)} T_i.size|T(s, e, h) \text{ is for } S(s, e) \text{ and } maxE(T(s, e, h)) \le p\}$$

Let MME and TE be as in Section 4.1 except that the terms are now for a fine-grain mapping onto an r-stage pipleine. It is easy to see that for $j \ge 0$,

$$MME(j, 1) = \begin{cases} 2^{j+1} & S(0, j) = \emptyset \\ \infty & \text{otherwise} \end{cases} \quad (14)$$

and for $r \ge 1$,

$$MME(0, r) = \max\{2, minMaxE(0, 0, r - 1)\} \quad (15)$$

Next, consider the case $j > 0$ and $r > 1$. Let $T(j, r)$ be as in Section 4.1. When the dest trie of the optimal pipelined 2DMT for levels 0 through j of O has only one level, $MME(j, r) = X(j, r)$, where

$$X(j, r) = \max\{2^{j+1}, minMaxE(0, j, r - 1)\}$$

When this dest trie has more than 1 level, the start-end pair for the last level is $(m+1, j)$ for some m, $0 \leq m < j$ and the last level (together with the source tries that hang from it) is mapped to s pipeline stages for some s, $1 \leq s < r$. So, $MME(j,r) = Y(j,r)$ where

$$Y(j,r) = \min_{0 \leq m < j, 1 \leq s < r} \{\max\{MME(m, r-s), nodes(m+1) * 2^{j-m}, minMaxE(m+1, j, s-1)\}\}$$

Combining the two cases for $MME(j,r)$, we get

$$MME(j,r) = \min\{X(j,r), Y(j,r)\}, j > 0, r > 1 \tag{16}$$

For TE, the following recurrence is obtained:

$$TE(j,1) = \begin{cases} 2^{j+1} & 2^{j+1} \leq MME(W-1,k) \text{ and } S(0,j) = \emptyset \\ \infty & otherwise \end{cases} \tag{17}$$

$$TE(0,r) = 2 + totalE(0, 0, r-1, MME(W-1,k)) \tag{18}$$

$$TE(j,r) = \min\{TX(j,r), TY(j,r)\}, j > 0, r > 1 \tag{19}$$

where

$$TX(j,r) = 2^{j+1} + totalE(0, j, r-1, MME(W-1,k))$$

$$TY(j,r) = \min_{m \in FX(j,r), 1 \leq s < r} \{TE(m, r-s) + nodes(m+1) * 2^{j-m} + totalE(m+1, j, s-1, MME(W-1,k))\}$$

and

$$FX(j,r) = \{m | 0 \leq m < j \text{ and } nodes(m+1) * 2^{j-m} \leq MME(W-1,k) \text{ and } minMaxE(m+1, j, s-1) \leq MME(W-1,k)\}$$

To compute $MME(W-1,k)$ and $TE(W-1,k)$ using the above dynamic programming equations, we must first compute $minMaxE$ and $totalE$. These quantities, however, appear to be difficult to compute. Instead, we estimate $minMaxE$ and $totalE$ using the algorithms for optimal pipelined FSTs [16] in case $T(s,e,h)$ is a set of FSTs and the heuristic of Section 3 in case $T(s,e,h)$ is a set of VSTs. For example, when $T(s,e,h)$ is a set of FSTs, we use the two-phase algorithm of [16] to determine, for each $S_i \in S(s,e)$, the FST U_i that has minimum total number of elements subject to the constraints that the FST has at most h levels and minimizes the maximum number of elements on any level. Let $levelSum(j)$ be the sum of the number of elements on level j of all the U_is. $minMaxE(s,e,h)$ is estimated to be $\max_j\{levelSum(j)\}$ and $totalE(s,e,h)$ is estimated to be the total number of elements in all the U_is.

Complexity Analysis

Prior to using Equations 14 through 16 to determine $MME(W-1,k)$, we must compute $O(W^2k)$ $minMaxE$ values. For this latter computation, we need to determine the approximately optimal pipelined VSTs (or optimal

pipelined FSTs) for all source tries $S(s, e)$. As noted in [19], the total number of source tries we need to work with is $O(nW)$ and each of these source tries has $O(nW)$ nodes. Using the heuristic of Section 3, the approximately optimal VSTs (for all possible heights) for these $O(nW)$ source tries may be computed in $O(n^2W^3k^2)$ time. If FST source tries are to be used, optimal FSTs for all possible heights and all possible source tries may be computed in $O(nW^3k)$ time using the algorithm of [16]. Each $minMaxE(s, e, h)$ is estimated to be $\max_j\{levelSum(j)\}$, where $levelSum(j)$ is the sume of the number of elements on level j of the approximately optimal (or optimal) pipelined VSTs (FSTs) that cover levels s through e of the 2D1BT. As $O(n)$ VSTs (or FSTs) are involved in the computation of each $levelSum(j)$ value and there are $O(k)$ levels to consider, an additional $O(nk)$ time is needed for each $minMaxE$ value that is to be computed. The additional time for all (s, e, h) combinations is, therefore, $O(nW^2k^2)$. (Note that the size of all of the VSTs (or FSTs) may be determined in this much time as well.) Adding up all the components, we see that the computation of all $minMax$ values takes $O(n^2W^3k^2)$ time in the case of VST source tries and $O(nW^3k + nW^2k^2)$ time in the case of FST source tries.

Once we have the $minMax$ values, Equations 14 through 16 may be solved for $MME(W-1, k)$ in $O(W^2k^2)$ time as there are $O(Wk)$ MME values to compute and each takes $O(Wk)$ time. Having computed $MME(W-1, k)$, we may proceed to compute $totalE(*, *, *, p)$ for $p = MME(W-1, k)$. $O(W^2k)$ $totalE$ values are to be computed; each taking $O(n)$ time. Thus all $totalE$ values may be computed in $O(nW^2k)$ time. Equations 17 through 19 may now be solved for $TE(W, k)$ in $O(W^2k^2)$ time ($O(Wk)$ TE values are to be computed at a cost of $O(Wk)$ time each).

So, the overall complexity of computing an approximately optimal pipelined 2DMT when the dest trie is an FST is $O(n^2W^3k^2)$ for the case of VST source tries and $O(nW^3k + nW^2k^2)$ for the case of FST source tries.

5.2 VST Dest Trie

Let T be a 2DMT in which the dest trie is a VST. Let E be an element field of any dest-trie node of T. Let $E.sourceTrie$ (equivalent to $E.data$ of Section 2.3) and $E.child$, respectively, denote the source subtrie and dest subtrie associated with E. We say that $E.sourceTrie$ and $E.child$ are *corresponding* tries. The 2DMT T may be mapped onto the stages of a pipeline architecture in the following way beginning with stage $nextStage$.

Step 1: The root M of T (but not the source tries that hang from it) is mapped to stage $nextStage$.

Step 2: Let $noCor(M)$ be the set of 2DMT subtries of M have no corresponding nonempty source tries. These 2DMT subtries are mapped recursively, one at a time, beginning at stage $nextStage + 1$.

Step 3: Let $S3(M)$ be the set of source tries that hang from M and that have no nonempty corresponding 2DMT subtrie. (Note that two or more elements of M may point to the same source trie S. If any one of these elements has a non-null *child* field, S is not in $S3(M)$.) The tries in $S3(M)$ are mapped level by level onto stages $nextStage+1$, $nextStage+2$, $\cdots$. Note that these source tries share pipeline stages among themselves as well as with the 2DMT subtries mapped in Step 2.

Filter	Dest	Source	Cost
F1	00*	0*	1
F2	000*	0*	2
F3	10*	000*	3
F4	111*	111**	4

Figure 9: A 4-filter database

Step 4: Let $S4(M)$ be the set of source tries that hang from M and that were not mapped in Step 3. Each of these source tries has one or more nonempty corresponding 2DMT subtrie. For $S \in S4(M)$, let $combo(S)$ $= (S, dest(S))$ be such that $dest(S)$ is the nonempty set of 2DMT subtries of M that correspond to S. The tries in each $combo(S)$ are mapped onto the pipeline beginning at stage $nextStage + 1$. For each $S \in S4(M)$, S is mapped level by level onto stages $nextStage + 1$, $nextStage + 2$, $\cdots$. Let l be the number of levels in S. The 2DMTs in $dest(S)$ are mapped one at a time and recursively beginning at stage $nextStage + l + 1$. Note that each $combo(S)$ shares pipeline stages with the source tries and 2DMT subtries mapped in Steps 2 and 3. Furher, different $combo(S)$s share stages among themselves.

As an example of this mapping strategy, consider the 4-filter database of Figure 9. Figure 10 shows a possible 2DMT for this database as well as the mapping of this 2DMT onto a pipeline. The mapping requires a 5-stage pipeline. The dest trie of this 2DMT is a VST and the source tries are FSTs. The mapping process begins with N being the root of the 2DMT and $nextStage = 1$. N is mapped to stage 1 in Step 1. N has 2 2DMT subtries, neither of which has a nonempty corresponding source trie. Both 2DMT subtries of N are mapped recursively beginning at stage 2 in Step 2. When N is the left child of the root, $nextStage = 2$. This left child is mapped to stage 2. The only source trie, S1, hanging from this left child and the only 2DMT subtrie of this left child define $combo(S1)$. $combo(S1)$ is mapped in Step 4 and so on. Notice that a pipeline stage may accomodate both dest- and source-trie nodes. So, we shall need to add a bit to each node to differentiate.

We develop dynamic programming recurrences to determine a 2DMT with VST dest and source tries that is suitable for fine grain mapping onto a k stage pipeline using the just stated mapping strategy. While the constructed 2DMT doesn't necessarily minimize the maximum per-stage memory requirement, it does produce good mappings. Let N be a dest-trie node in the 2D1BT O. We use the following additional terminology:

$Opt'(N, r) \cdots$ This is the approximately optimal pipelined 2DMT for the subtrie $ST(N)$ such that the 2DMT maps onto at most r pipeline stages using the just stated mapping strategy. Let M be the root of $Opt'(N, r)$. In $Opt'(N, r)$, the subtries of M are themselves approximately optimal 2DMTs and the source tries are approximately optimal VSTs (as defined in Section 3).

$Opt'(N, r).E(l) \cdots$ This is the number of elements mapped to stage l of the pipeline.

$D'_{q+1}(N) \cdots$ This is the subset of $D_{q+1}(N)$ that are the roots of the subtries represented by $noCor(M)$ under the assumption that M covers levels 1 through q of $ST(N)$.

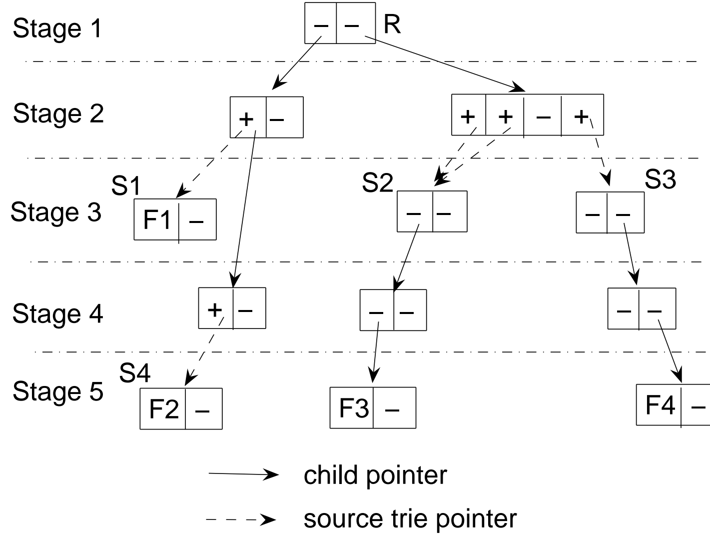


Figure 10: A 2DMT for Figure 9

$D''_{q+1}(N) \cdots$ $D''_{q+1}(N) = D_{q+1}(N) - D'_{q+1}(N)$

$Tries(N, q, z) \cdots$ This is the set of approximately optimal pipelined VSTs for $S3(M)$. The pipelined VSTs have at most z levels each and M covers levels 0 through q of $ST(N)$.

$Tries(N, q, z).E(l) \cdots$ This is the total number of elements at level l of all VSTs in $Tries(N, q, z)$.

$OptCombo'(S, N, q, z).E(l) \cdots$ This gives the number of elements assigned to stage $l + 1$ of the pipeline when $combo(S)$ is mapped to a pipeline beginning with stage 1; the root M of the 2DMT for $ST(N)$ covers levels 0 through q of $ST(N)$; and the mapping of $combo(S)$ is limited to z stages.

From these definitions and the fine-grain mapping strategy, it follows that if M covers levels 0 through q of $ST(N)$ then the maximum number of elements assigned to any stage of a z-stage pipeline is

$$ME(N, q, z) = \max\{2^{q+1}, h(N, q, z - 1)\}$$

where

$$h(N, q, z) = \max_{l \in \{0..z-1\}} \{h(N, q, z, l)\}$$

and

$$h(N, q, z, l) = \sum_{R \in D'_{q+1}(N)} Opt'(R, z).E(l) + Tries(N, q, z).E(l) + \sum_{S \in S4(M)} OptCombo'(S, N, q, z).E(l)$$

For given N, q and z, let

$$X = argmin_q\{ME(N, q, z)\}$$

We see that

$$Opt'(N,r).E(l) = \begin{cases} 2^{X+1} & l = 0 \\ h(N,X,r-1,l-1) & 0 < l < r \end{cases} \quad (20)$$

To obtain the recurrence for $OptCombo'$, we define the function $g(S,N,q,z,y)$, which gives the maximum per-stage number of elements when $combo(S)$ is mapped using an approximately optimal VST, $Opt'(S,y)$, for $S \in S4(M)$; the VST has at most y levels; the subtries in $dest(S)$ are approximately optimal 2DMTs whose mapping onto a pipeline requires at most $z-y$ stages; and M is a multibit dest trie node that covers levels 0 through q of $ST(N)$. From our mapping stratgey, it follows that

$$g(S,N,q,z,y) = max\{\max_{l \in \{0..y-1\}}\{Opt'(S,y).E(l)\}, \max_{l \in \{0..z-y-1\}}\{\sum_{R \in D''_{q+1}(N)} Opt'(R,z-y).E(l)\}\}$$

For given S, N, q and z, let

$$Y = argmin_{1 \le y < z}\{g(S,N,q,z,y)\}$$

The recurrence for $OptCombo'$ is:

$$OptCombo'(S,N,q,z).E(l) = \begin{cases} Opt'(S,Y).E(l) & 0 \le l < Y \\ \sum_{R \in D''_{q+1}(N)} Opt'(R,z-Y).E(l-Y) & Y \le l < z \end{cases} \quad (21)$$

Complexity Analysis

As in the case when the dest trie was an FST, we start by computing either approximately optimal pipelined VSTs or optimal FSTs (of all possible heights) for all source tries that may hang off of multibit dest-trie nodes. The cost for this is $O(n^2W^3k^2)$ in the case of VST source tries and $O(nW^3k)$ in the case of FST source tries. There are $O(nW^2k^2)$ $Tries(*,*,*).E(*)$ values to compute. As the number of source tries that can hang off of each multibit node M is $O(n)$, each $Tries(N,q,z).E(l)$ value may be computed in $O(n)$ time from the already computed source VSTs (or FSTs). Hence, all $Tries(*,*,*).E(*)$ values may be computed in $O(n^2W^2k^2)$ time. For $OptCombo'$, we see that for any triple (N,q,z), there are only $O(n)$ possible S values. Hence, there are only $O(n^2W^2k^2)$ $OptCombo'(*,*,*,*).E(*)$ values to compute. Let $Smax$, $Nsum$ and $Nmax$ be as below:

$$Smax(S,y) = \max_{l \in \{0..y-1\}}\{Opt'(S,y).E(l)\}$$

$$Nsum(N,q,z,l) = \sum_{R \in D''_{q+1}(N)} Opt'(R,z).E(l)$$

$$Nmax(N,q,z,y) = Nmax(N,q,z-y) = \max_{l \in \{0..z-y-1\}}\{Nsum(N,q,z-y,l\}$$

As there are only $O(nW)$ S values and $O(k)$ y values, all $Smax$ values may be computed from the already computed Opt' values in $O(nWk^2)$ time. The total number of $Nsum$ values is $O(nW^2k^2)$. Since each node of the dest trie of the 2D1BT has $O(W)$ ancestors, each of the $O(nW)$ nodes in this dest trie can be involved in the computation of only $O(W)$ of the $O(nW^2)$ $Nsum$ values for any given pair (z,l). Hence, for any given (z,l), all

$Nsum(*,*,z,l)$ values may be computed in $O(nW^2)$ time. As there are $O(k^2)$ (z,l) pairs, the time to compute all $Nsum$ values is $O(nW^2k^2)$. From the computed $NSum$ values, all $O(nW^2k)$ $Nmax(*,*,*)$ values may be determined in an additional $O(n^2W^2k^2)$ time. Now, each $Nmax(N,q,z,y)$ value is determined in $O(1)$ time as it is equal to $Nmax(N,q,z-y)$. From the $Smax$ and $Nmax$ values, all $O(n^2W^2k^2)$ g values may be computed in $O(n^2W^2k^2)$ time. Note that

$$g(S,N,q,z,y) = \max\{Smax(S,y), Nmax(N,q,z,y)\}$$

Now, for each (S,N,q,z), we can determine Y in $O(k)$ time and from Y compute $OptCombo'(S,N,q,z).E(l)$ (Equation 21) in $O(1)$ time. So, all $O(n^2W^2k^2)$ $OptCombo'(S,N,q,z).E(l)$ values may be computed in $O(n^2W^2k^2)$ additional time. Since each of the $O(nW)$ nodes in the dest trie of the 2D1BT has $O(W)$ ancestors and $|S4(M)| \leq n$ for every M, all $O(nW^2k^2)$ h values and hence, all $Opt'(*,*).E(*)$ values, may be computed in $O(n^2W^2k^2)$ time.

Adding together all components of the time complexity, we see that an approximately optimal fine-grain pipelined 2DMT with a VST dest trie may be computed in $O(n^2W^3k^2)$ time when the source tries are also VSTs and in $O(nW^3k+n^2W^2k^2) = O(n^2W^2k^2)$ (under the very realistic assumption that $W = O(nk)$) time when the source tries are FSTs.

6 Experimental Results

Our algorithms for one- and two-dimensional pipelines multibit tries were programmed in C++ and compiled using the GCC 3.3.5 compiler with optimization level `O3`. The compiled codes were run a 2.80 GHz Pentium 4 PC. Our algorithms for pipelined one-dimensional multibit tries were benchmarked against the best one-dimensional multibit trie algorithms of [16] and out two-dimensional algorithms were benchmarked against the two-dimensional algorithms of [19]. In the following experiments, two optimization techniques *packed array* [30] and *butler node* [18] are employed to compress one- and two-dimensional tries. These two techniques are very similar; both attempt to replace a subtrie with a small amount of actual data (prefixes and pointers) by a single node that contains these data.

6.1 Pipelined One-dimensional Multibit Tries

Basu and Narliker [4] and Kim and Sahni [16] have proposed algorithms for the construction of pipelined one-dimensional multibit tries. Since the algorithms of [16] are superior to those of [4], we focus on the algorithms of [16]. [16] develops an algorithm PFST-2, which is a 2-stage algorithm that results in pipelined FSTs that miminize total memory subject to minimizing the maximum per-stage memory. Sahni and Kim [16] also propose two algorithms PU-2n and PART that are based on FSTs but result in VSTs that are superior for pipeline applications than the optimal FSTs generated by PFST-2. In particular, the PU-2n tries require smaller total memory than do the tries of PFST-2; the maximum per-stage memory no more than that for PFST-2. The PART tries have a smaller maximum per-stage memory requirement than the tries of PFST-2 but require more total memory. Henceforth, we abbreviate PFST-2 to PFST. Let VST denote the algorithm of Sahni and Kim [25], which constructs VSTs

with minimum total memory and let PVST be our algorithm of Section 3. So, in all, we have 5 algorithms–PFST, PU-2n, PART, VST and PVST–for the construction of pipelined multibit tries. Only one of these, PFST, results in an FST and the others result in VSTs.

We first determine the effectiveness of our tree packing heuristic of Figure 6 relative to the straightforward mapping (i.e., nodes at level l of the multibit trie are packed into stage $l+1$, $0 \leq l < k$ of the pipeline). For our experiment, we use the six IPv4 router tables Aads, MaeWest, RRC01, RRC04, AS4637 and AS1221 that were obtained from [21, 23, 14]. The number of prefixes in these router tables is 17486, 29608, 103555, 109600, 173501 and 215487, respectively. Table 1 gives the reduction in maximum per-stage memory when we use our tree packing heuristic rather than the straightforward mapping. For example, on our 6 data sets, the tree packing heuristic reduced the maximum per-stage memory required by the multibit trie generated by PVST by between 0% and 38%; the mean reduction was 14% and the standard deviation was 13%. The reduction obtained by the tree packing heuristic was as high as 60% when applied to the tries constructed by the algorithms of [16].

Algorithm	Min	Max	Mean	Standard Deviation
PFST	0%	60%	33%	22%
PU-2n	0%	60%	33%	22%
PART	0%	59%	25%	20%
VST	0%	27%	9%	10%
PVST	0%	38%	14%	13%

Table 1: Reduction in maximum per-stage memory resulting from tree packing heuristic

Tables 2 and 3, respectively, give the maximum per-stage memory and total memory requirements for the multibit tries resulting from our 5 algorithms. In each case, the tries were mapped into k, $2 \leq k \leq 8$, pipeline stages using our tree packing heuristic. Figure 11 plots this data for AS1221.

k		2	3	4	5	6	7	8
Aads	PFST	2304	216	118	64	64	64	64
	PU-2n	2304	216	118	64	64	64	64
	PART	2304	219	105	72	71	51	49
	VST	576	94	60	42	33	30	27
	PVST	576	88	46	36	26	23	18
MaeWest	PFST	7634	327	144	102	102	102	102
	PU-2n	7634	327	144	102	102	102	102
	PART	5838	369	151	144	94	78	72
	VST	896	175	97	71	59	52	44
	PVST	896	144	73	55	42	37	33
RRC01	PFST	9216	708	362	308	308	230	230
	PU-2n	9216	708	360	301	301	225	225
	PART	9216	708	380	288	324	248	188
	VST	1866	587	335	229	192	161	153
	PVST	1866	447	248	175	146	114	111
RRC04	PFST	10490	1048	395	339	243	243	243
	PU-2n	10490	1048	394	334	239	239	239
	PART	10490	1048	445	326	288	261	209
	VST	2304	504	349	240	207	169	154
	PVST	2304	482	264	187	155	121	118
AS4637	PFST	18432	1026	560	479	479	479	359
	PU-2n	18432	1026	557	469	469	469	351
	PART	15149	989	576	406	368	358	315
	VST	2517	751	555	376	321	268	245
	PVST	2304	597	368	288	288	185	181
AS1221	PFST	18432	1714	795	498	439	439	348
	PU-2n	18432	1714	795	496	431	431	342
	PART	18432	1743	818	576	387	365	311
	VST	5750	1002	704	497	404	324	298
	PVST	4608	802	476	332	313	288	217

Table 2: Maximum per-stage memory (KB)

k		2	3	4	5	6	7	8
Aads	PFST	3715	577	350	242	230	230	230
	PU-2n	3715	577	350	242	230	230	230
	PART	3482	566	355	330	326	297	300
	VST	1010	238	140	121	117	116	116
	PVST	1010	242	163	157	139	132	132
MaeWest	PFST	12242	879	540	367	356	355	355
	PU-2n	12242	879	540	367	355	355	355
	PART	10446	965	599	575	477	469	469
	VST	1472	368	232	202	196	194	194
	PVST	1472	402	282	239	239	221	216
RRC01	PFST	18049	1920	1375	1062	1039	1035	1034
	PU-2n	18049	1920	1370	1042	1018	1015	1013
	PART	17050	1920	1429	1440	1528	1524	1314
	VST	3018	957	680	626	613	610	609
	PVST	3018	980	815	799	712	715	671
RRC04	PFST	19706	2283	1475	1159	1114	1090	1088
	PU-2n	19706	2283	1471	1142	1097	1073	1071
	PART	19706	2283	1623	1590	1553	1605	1463
	VST	4440	1039	726	663	649	645	643
	PVST	4440	1057	854	835	748	751	706
AS4637	PFST	19445	2628	1968	1624	1595	1594	1593
	PU-2n	19445	2628	1961	1594	1565	1563	1562
	PART	24365	2428	2214	1913	1784	2151	2150
	VST	3669	1371	1071	1003	986	981	980
	PVST	3801	1565	1248	1253	1260	1101	1065
AS1221	PFST	23161	4580	2664	2283	1939	1911	1896
	PU-2n	23161	4580	2664	2272	1906	1878	1864
	PART	23161	4629	2929	2650	2227	2264	2250
	VST	8054	1730	1313	1217	1195	1189	1187
	PVST	8403	1913	1520	1485	1361	1485	1308

Table 3: Maximum per-stage memory (KB)

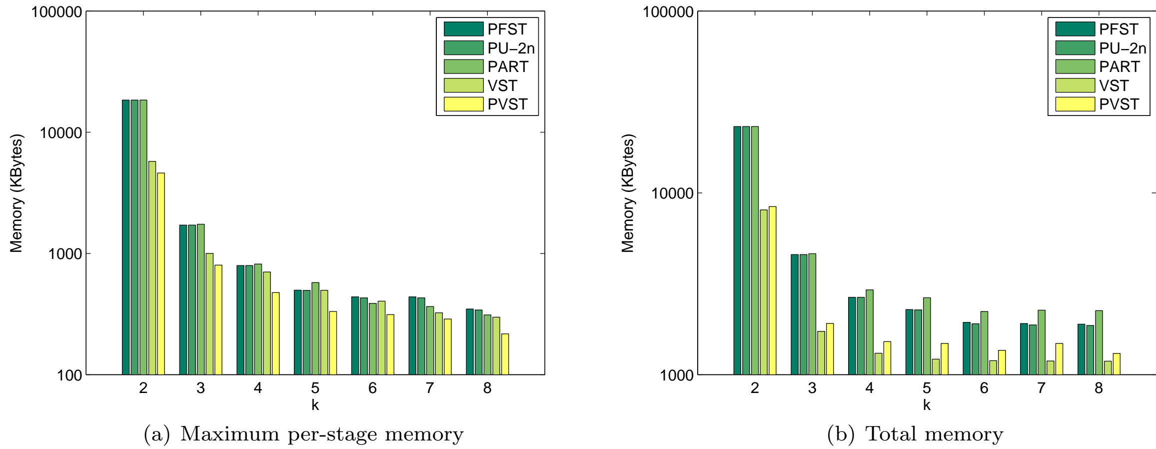


Figure 11: Maximum per-stage and total memory (KB) for AS1221

In all of our 42 tests, PVST results in the least maximum per-stage memory requirement. Tables 4 and 5 give the maximum per-stage and total memory required by the 5 algorithms normalized by the requirements for PVST. The maximum per-stage memory requirement for the algorithms of [16] are up to 8.51 times that of PVST while the requirement for VST is up to 50% more than that of PVST. On average, the total memory required by the multibit tries produced by VST was 12% less than that required by the PVST tries; the tries generated by the algorithms of [16] required, on average, about 2 times the total memory required by the PVST tries.

Algorithm	Min	Max	Mean	Standard Deviation
PFST	1.40	8.51	2.54	1.54
PU-2n	1.38	8.51	2.53	1.551
PART	1.24	6.58	2.38	1.25
VST	1.00	1.50	1.28	0.14
PVST	1	1	1	0

Table 4: Maximum per-stage memory normalized by PVST's maximum per-stage memory

Algorithm	Min	Max	Mean	Standard Deviation
PFST	1.27	8.31	2.13	1.41
PU-2n	1.24	8.31	2.12	1.41
PART	1.42	7.09	2.42	1.23
VST	0.77	1	0.88	0.07
PVST	1	1	1	0

Table 5: Total memory normalized by PVST's total memory

6.2 Pipelined Two-dimensional Multibit Tries

We benchmarked our coarse and fine grain two-dimensional multibit trie algorithms against the heuristics of Lu and Sahni [19] for two-dimensional multibit tries that minimize total memory. Lu and Sahni [19] propose 4 heuristics–2DMTa through 2DMTd–to construct two-dimensional multibit tries. Of these, only 2DMTa constructs tries suitable for coarse mapping[5]. Although 2DMTd is the best of the heuristics of [19] (from the standpoint of total memory requirement), the tries constructed by 2DMTd are suited only for a fine mapping on to a pipeline. So, we compare our coarse grain algorithms to 2DMTa and our fine grain algorithms to 2DMTd. For test data, we used the 12 data sets of [19]. These data sets are derived from 5-dimensional data sets generated using the filter generator of [33]. Each of these data sets actually has 10 different databases of filters. So, in all, we have 120 databases of two-dimensional filters. The data sets, which are named ACL1 through ACL5, FW1 through FW5, IPC1 and IPC2 have, respectively, 20K, 19K, 10K, 13K, 5K, 19K, 19K, 18K, 17K, 17K, 16K and 20K filters, on average, in each database.

6.2.1 Coarse Pipeline Mapping

As was the case for one-dimensional tries, the maximum per-stage memory depends both on the trie that is mapped to the pipelined architecture and the mapping strategy used. Table 6 gives the reduction in maximum per-stage memory when the tree packing heuristic of Figure 6 is used versus the straightforward mapping. Although the mean reduction (less than 7%) is small, the maximum reduction (51%) is significant. All remaining data presented in this section are for the case when the tree packing heuristic of Figure 6 is used.

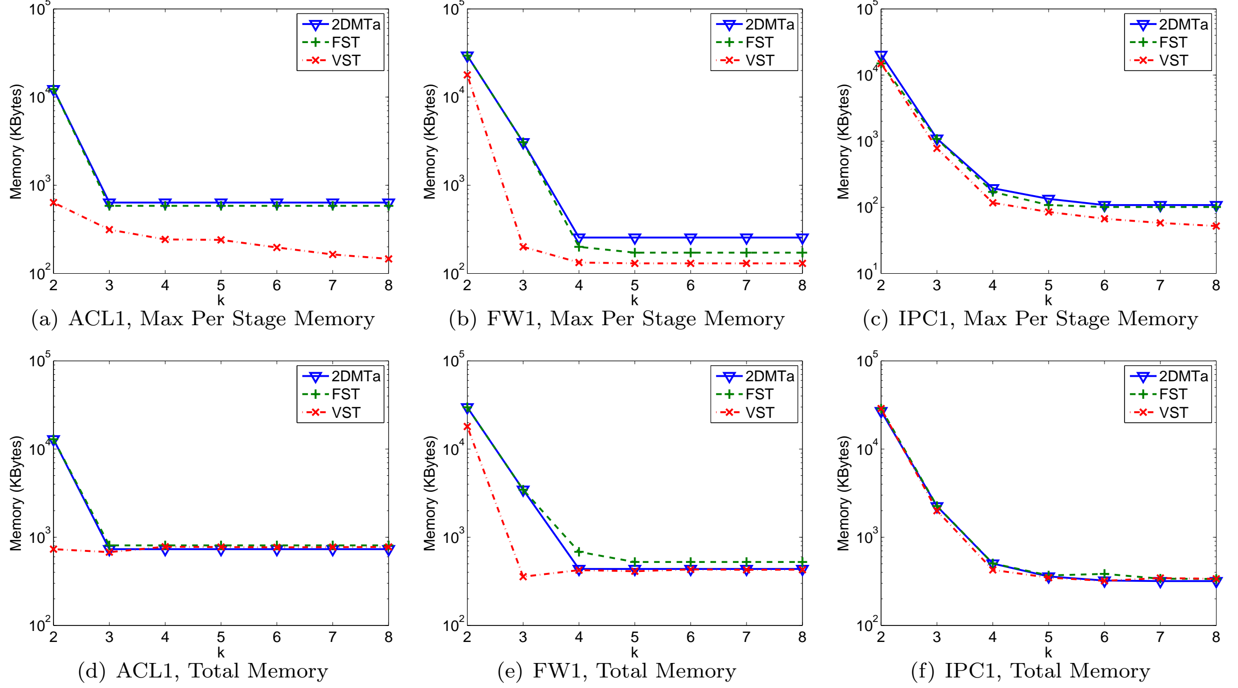
Algorithm	Min	Max	Mean	Standard Deviation
2DMTa	0%	19%	2%	5%
FST	0%	19%	2%	5%
VST	0%	51%	7%	12%

Table 6: Reduction in maximum per-stage memory resulting from tree packing heuristic

In our experiments, the source tries were restricted to have height at most $maxSH$ for $maxSH \in \{2, 3, 4\}$ and the dest-trie height was restricted to be at most k, $2 \leq k \leq 8$. With these restrictions, the constructed two-dimensional trie could be mapped into a k-stage pipeline with a cycle time of $maxSH + 1$ (1 to access a dest-trie node and $maxSH$ to access source-trie nodes). For FST, the tree-packing heuristic of Figure 3.2 was employed to reduce the maximum per-stage memory and for VST, the node pullup scheme of Section 4.2 followed by the tree-packing heuristic of Figure 6 were employed. The node pullup scheme reduced the maximum per-stage memory by as much as 51% (the minimum and mean reductions were 0% and 3%, respectively; the standard deviation was 11%) and the tree-packing heuristic reduced this by an additional up to 19% for FSTs and up to 51% on VSTs (as reported in Table 6). Figure 12 shows the maximum per-stage and total memory requirements

[5]Note that for good pipeline performance, the time spent in each stage of the pipeline should be approximately the same. In the context of a coarse mapping, this means that the height of all source tries should be (approximately) equal.

of the coarse grain mapping resulting from the tries produced by 2DMTa and the FST and VST dest-trie coarse mapping algorithms of Section 4 for the data sets ACL1, FW1 and IPC1 and the case $maxSH = 4$. Since each data set is comprised of 10 databases, we actually show the average of the 10 values. The experimental results are similar for the remaining 9 data sets.



(a) ACL1, Max Per Stage Memory (b) FW1, Max Per Stage Memory (c) IPC1, Max Per Stage Memory

(d) ACL1, Total Memory (e) FW1, Total Memory (f) IPC1, Total Memory

Figure 12: Maximum per-stage and total memory required by coarse-grain pipelining, $maxSH = 4$. Source tries are VSTs.

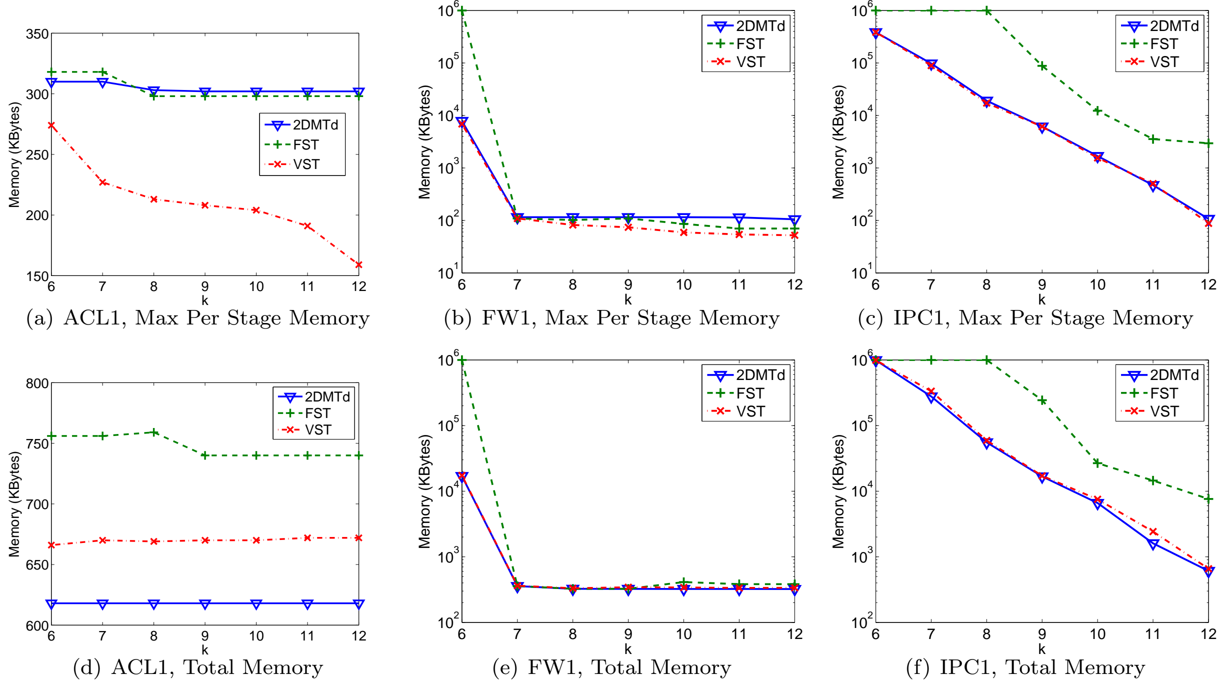
On our test data, the maximum per-stage memory and total memory of VST are generally superior to that of FST and to that of 2DMTa. The maximum per-stage memory of FST normalized by that of VST is between 0.91 and 80; the mean and standard deviation are 3.16 and 5.95, respectively. The normalized numbers for total memory required are between 0.58 and 65 with the mean and standard deviation being 2.04 and 4.80. On our test data, the maximum per-stage memory of 2DMTa normalized by that of VST is between 0.93 and 80; the mean and standard deviation are 3.39 and 5.91, respectively. The normalized numbers for total memory required are between 0.49 and 65 with the mean and standard deviation being 1.92 and 4.74. For the structures constructed by these three algorithms, increasing $maxSH$ significantly reduced the per-stage (total) memory requirement on 4 of our databases (ACL3-5 and IPC1), while has no or slight impact on the remaining 8 databases.

6.2.2 Fine Pipeline Mapping

The tree packing heuristic had a very small impact (less than 1%) on our fine-grain pipeline algorithms. Our remaining results are for the case when the tree packing heuristic is used.

Figure 13 shows the maximum per-stage and total memory requirements of the fine grain mapping resulting

from the tries produced by 2DMTd(k) and the FST and VST dest-trie fine mapping algorithms of Section 5. Memory requirements in excess of 10^6 are plotted as 10^6. Node pullup did not reduce the maximum per-stage memory requirement of the trie resulting from our VST dest-trie fine mapping algorithm.



(a) ACL1, Max Per Stage Memory (b) FW1, Max Per Stage Memory (c) IPC1, Max Per Stage Memory

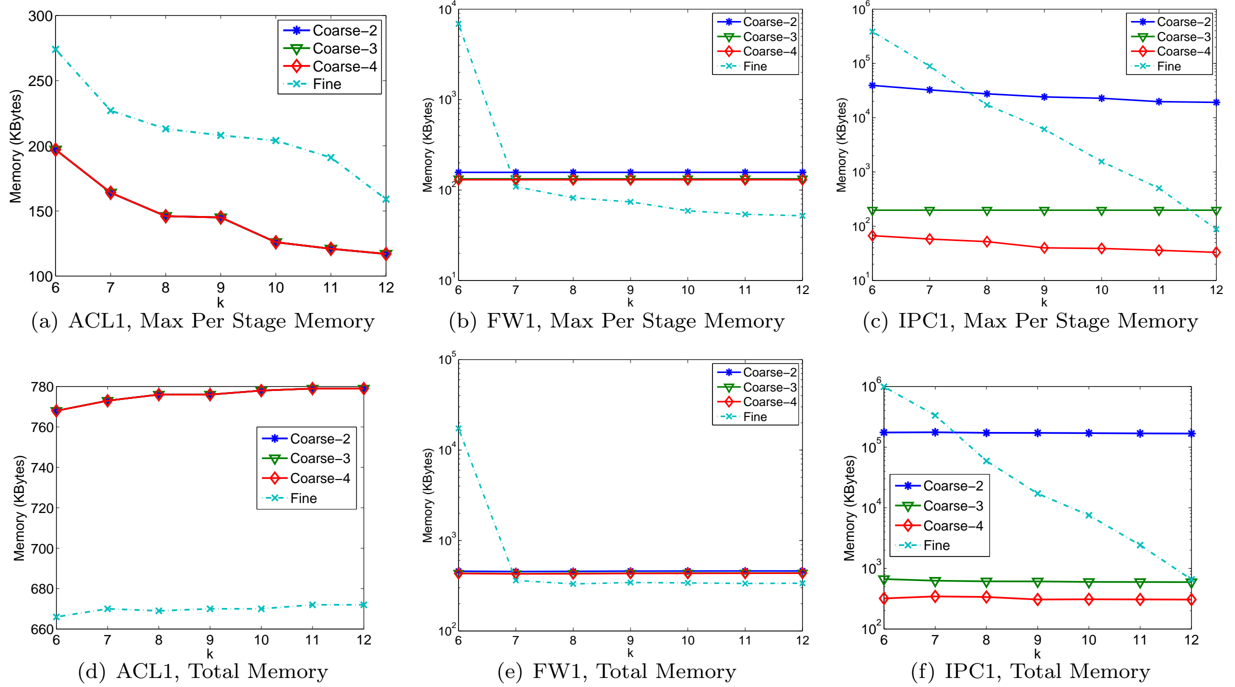
(d) ACL1, Total Memory (e) FW1, Total Memory (f) IPC1, Total Memory

Figure 13: Maximum per-stage and total memory required by fine-grain pipelining. Source tries are VSTs.

For all our data sets, VST generally outperformed FST on both the maximum per-stage metric as well as the total memory metric. In fact, the maximum per-stage memory for FST normalized by that for VST was between 0.7 and 2047; the mean and standard deviation were 109 and 328, respectively. The normalized numbers for total memory were between 0.82 and 1020. The mean and standard deviation were 42 and 128. VST was also generally superior to 2DMTd on the maximum per-stage memory metric. The maximum per-stage memory for 2DMTd normalized by that for VST was between 0.65 and 2.16; the mean and standard deviation were 1.27 and 0.29, respectively. Not surprisingly, 2DMTd was always superior to VST on the total memory metric. The normalized numbers for total memory were between 0.40 and 1.00 and the mean and standard deviation were 0.91 and 0.07.

Figure 14 compares the memory requirements of the tries resulting from the best coarse grain and fine grain algorithms (i.e., the VST dest-trie algorithms of Sections 4 and 5). For the case of a coarse grain mapping, we show three cases–$maxSH = 2$ (Coarse-2), 3 and 4. The fine grain mapping is superior to the coarse grain mapping when k is large but not when k is small. For FW1, for example, the maximum per-stage memory required by the coarse ($maxSH = 3$) and fine mapping is (almost) the same when $k = 7$. However, the cycle time for the 7-stage pipeline is 4 for the coarse mapping and 1 for the fine mapping. So, the fine mapping results in a throughput that is 4 times that of the coarse mapping while using (almost) the same maximum per-stage memory. On the other

hand, when $k = 6$, the fine mapping required up to 6050 times the maximum per-stage memory required by a coarse mapping with $maxSH = 4$! When $k = 12$, the coarse mapping with $maxSH = 2$ required up to 1061 times the maximum per-stage memory required by the fine mapping.



(a) ACL1, Max Per Stage Memory (b) FW1, Max Per Stage Memory (c) IPC1, Max Per Stage Memory

(d) ACL1, Total Memory (e) FW1, Total Memory (f) IPC1, Total Memory

Figure 14: Maximum per-stage and total memory required by coarse-grain and fine-grain pipelining. The dest trie is VST and the source tries are VSTs.

7 Conclusion

We have developed a tree packing heuritic that reduces the maximum per-stage memory required by the one-dimensional multibit tries of [16] by as much as 60%. Our PVST algorithm results in VST multibit tries, which when mapped into a pipelined architecture, have a maximum per-stage memory requirement that is up to 1/8 that required by the tries of [16].

For two-dimensional multibit tries, we have proposed two strategies–fine and coarse–for mapping into a pipeline. For each strategy, we have developed dynamic programming algorithms for both FST and VST dest and source tries. Although these algorithms do not find optimal multibit tries, they construct coarse grain multibit tries that generally have a much smaller maximum per-stage memory requirement than do the tries obtained using the 2DMTa algorithm of [19] and they construct fine grain multibit tries that generally have a much smaller maximum per-stage memory requirement than do the tries obtained using the 2DMTd algorithm of [19]

References

[1] F.Baboescu and G.Varghese, Scalable packet classification, *ACM SIGCOMM*, 2001.

[2] F.Baboescu and G.Varghese, Fast and Scalable Conflict Detection for Packet Classifiers, *10th IEEE International Conference on Network Protocols (ICNP'02)*,2002.

[3] F.Baboescu, S.Singh and G.Varghese, Packet Classification for Core Routers: Is there an alternative to CAMs? *INFOCOM*, 2003.

[4] A. Basu and G. Narlikar Fast Incremental Updates for Pipeline Forwarding Engines, *InfoCom*, 2003.

[5] J.L.Bentley and T.A.Ottmann, Algorithms for Reporting and Counting Geometric Intersections, *IEEE Transactions on Computers*, C-28, 9, 1979, 643-647.

[6] M. Buddhikot, S. Suri and M. Waldvogel, Space decomposition techniques for fast layer-4 switching, *Conference on High Speed Networks*, 1998.

[7] D. Eppstein and S. Muthukrishnan, Internet packet filter management and rectangle geometry, *12th ACM-SIAM Symp. on Discrete Algorithms*, 2001, 827-835.

[8] A. Feldman and S. Muthukrishnan, Tradeoffs for packet classification, *INFOCOM*, 2000.

[9] P. Gupta and N. McKeown, Packet classification using hierarchical intelligent cuts, *ACM SIGCOMM*, 1999.

[10] H. Lu and S. Sahni, O(log W) multidimensional packet classification, paper in review.

[11] H. Lu and S. Sahni, Conflict detection and resolution in two-dimensional prefix router tables, *IEEE/ACM Transactions on Networking*, to appear.

[12] A.Hari, S.Suri, G.Parulkar, Detecting and resolving packet filter conflicts, *INFOCOM*, 2000.

[13] E.Horowitz, S.Sahni, and S.Rajasekeran, Computer Algorithms/C++, W. H. Freeman, NY, 1997.

[14] *http://www.merit.edu/ipma/routing_table*

[15] C. Kaufman, R. Perlman and M. Speciner, *Network Security: Private communication in a public world*, Second Edition, Chapter 17, Prentice Hall, NJ, 2002.

[16] K. Kim and S. Sahni, Efficient construction of pipelined multibit-trie Router-Tables, *Paper in Review.*

[17] T. Lakshman and D. Stidialis, High speed policy-based packet forwarding using efficient multi-dimensional range matching, *ACM SIGCOMM*, 1998.

[18] B. Lampson, V. Srinivasan, and G. Varghese, IP Lookup using Multi-way and Multi-column Search, *IEEE Infocom 98*, 1998.

[19] W. Lu and S. Sahni, Packet Classification Using Two-Dimensional Multibit Tries, *IEEE Symposium on Computers and Communications*, 2005.

[20] C. W. Mortensen, Fully-dynamic two dimensional orthogonal range and line segment intersection reporting in logarithmic time, *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, 2003.

[21] *http://bgp.potaroo.net*

[22] L. Qiu, G. Varghese and S. Suri, Fast firewall implementation for software and hardware based routers. *9th International Conference on Network Protocolos ICNP*, 2001.

[23] Ris, Routing information service raw data, *http://data.ris.ripe.net*

[24] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous, Survey and taxonomy of IP address lookup algorithms, *IEEE Network*, 2001, 8-23.

[25] S. Sahni and K. Kim, Efficient construction of multibit tries for IP lookup, *IEEE/ACM Transactions on Networking*, 11, 4, 2003.

[26] S. Sahni, K. Kim, and H. Lu, Data structures for one-dimensional packet classification using most-specific-rule matching, *International Journal on Foundations of Computer Science*, 14, 3, 2003, 337-358.

[27] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, Fast and scalable layer four switching, *Proc. ACM SIGCOMM*, 1998.

[28] V. Srinivasan, S. Suri, and G. Varghese, Packet classification using tuple space search, *ACM SIGCOMM*, 1999.

[29] V. Srinivasan, A packet classification and filter management system, *INFOCOM*, 2001.

[30] V. Srinivasan and G. Varghese, Faster IP lookups using controlled prefix expansion, *ACM Transactions on Computer Systems*, Feb:1-40, 1999.

[31] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, Scalable Algorithms for Layer four Switching, *Proceedings of ACM Sigcomm*, 8, 1998.

[32] X.Sun and Y.Zhao, Packet classification using independent sets, *IEEE Symposium on Computers & Communications*, 2003, 83-90.

[33] David E. Taylor, Jonathan S. Turner, ClassBench: A Packet Classification Benchmark, *INFOCOM*, 2005.