Packet Classification Using Pipelined Two-Dimensional 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

October 14, 2005

Abstract

We propose heuristics for the construction of fixed- and variable-stride two-dimensional multibit tries. These multibit tries are suitable for the classification of Internet packets using a pipelined architecture. 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 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.

Data structures for multi-dimensional (i.e., $d > 1$) packet classification are surveyed in [1, 2, 4, 7]. In this paper, we focus on the development of data structures suitable for ASIC-based pipelined architectures for high speed two-dimensional packet classification. Basu and Narlikar [10] and Kim and Sahni [6] 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 [10] 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.

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

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

Basu and Narliker [10] assert that constraint C3 reduces pipeline disruption resulting from rule-table updates. Although the algorithm proposed in [10] constructs fixed-stride tries that satisfy constraints C1 and C2, the constructed tries may violate constraint C3. Kim and Sahni [6] 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 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 Lu and Sahni [8]. 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 [8] 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 [3], and two-dimensional 1- and multi-bit tries [8]. 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 3 and those for fine-grain mapping in Section 4. An experimental evaluation of our algorithms is conducted in Section 5.

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.

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.

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.

Starting with a 1-bit trie for n prefixes whose length is at most W, the strides for a space-optimal FST with at most k levels may be determined in $O(nW + kW^2)$ time[1] [3, 5]. For a space-optimal VST whose height[2] is constrained to k, the strides may be determined in $O(nW^2k)$ time [3, 5].

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 1. 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 1 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 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 2(a) gives the 2D1BT for the filters of Figure 1.

2.4 Two-Dimensional Multibit Tries

Two-dimensional multibit tries (2DMTs) [8] 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

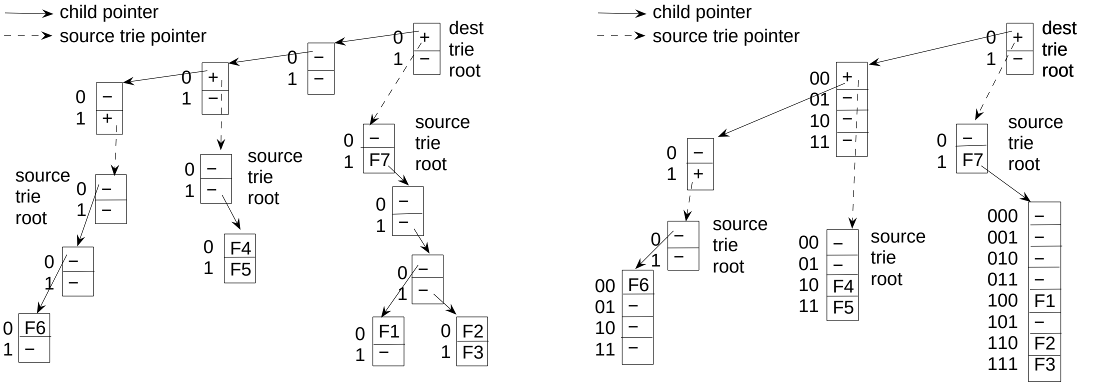
[1] The complexity of $O(kW^2)$ given in [3, 5] assumes we start with data extracted from the 1-bit trie; the extraction of this data takes $O(nW)$ time.

[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 1: An example of seven dest-source filters



(a) 2D1BT

(b) 2DMT

Figure 2: Two-dimensional 1- and multi-bit trie for Figure 1

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 1, $u = 3$, $D_1 = 0*$, $D_2 = 000*$ 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 2(b)). 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 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 2(b), 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 2(b) 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 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 2(b) 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 2(b) 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 4. 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 5 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.

3.1 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

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

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 [8]. Then, $O(kW)$ MME values are computed using Equation 1. 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

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

For $r > 1$, we get

$$TE(j,r) = \min_{m \in X(j,r)} \max\{TE(m, r-1) + nodes(m+1) * 2^{j-m} + sourceSum(m+1,j)\}\} \tag{3}$$

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

The additional time needed to compute $TE(W-1,k)$ using Equations 2 and 3 is $O(kW^2)$.

3.2 VST Dest Trie

We proposed a heuristic in this section 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 $Opt'(N,r)$, $0 \le l < r$ (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 $ST(N)$. 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,

$$Opt'(N,1).E(l) = \begin{cases} 2^{h(N)} + sourceSum(N, h(N)-1) & l = 0 \\ 0 & l > 0 \end{cases} \tag{4}$$

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

$$Opt'(N,r).E(l) = \begin{cases} 2^q + sourceSum(N, q-1) & l = 0 \\ \sum_{M \in D_q(N)} Opt'(M, r-1).E(l-1) & 1 \leq l < r \end{cases} \tag{5}$$

where q is as defined below

$$q = argmin_{1 \leq i \leq h(N)} \{\max\{2^i + sourceSum(N, i-1), \max_{0 \leq l < r-1} \{ \sum_{M \in D_i(N)} Opt'(M, r-1).E(l)\}\}\} \tag{6}$$

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 $Opt'STs(N, i, r-1)$. So,

$$Opt'STs(N, i, r-1).E(l) = \sum_{M \in D_i(N)} Opt'(M, r-1).E(l), 0 \leq l < r-1 \tag{7}$$

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

$$Opt'STs(N,i,r-1).E(l) = \begin{cases} Opt'(LC(N), r-1).E(l) + Opt'(RC(N), r-1).E(l) & i = 1 \\ Opt'STs(LC(N), i-1, r-1).E(l) + Opt'STs(RC(N), i-1, r-1).E(l) & i > 1 \end{cases} \tag{8}$$

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

All values of $sourceSum(N, j)$ may be computed in $O(n^2W^3k)$ time as in [8]. Using Equations 4–8, 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 4–8 by employing the node pullup technique [10, 6]. 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 3. Suppose that the optimal H-level VST for each of the source tries $S1$ and $S2$ has 100 elements. When Equations 4–8 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

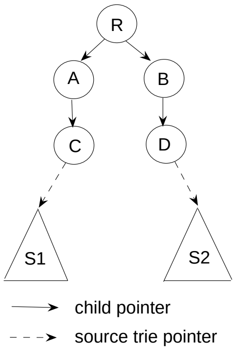


Figure 3: Example for node pullup

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 approximately optimal j-level 2DMT whose maximum per-stage memory requirement is $Opt'(M, j)$. Algorithm *NodePullup* (Figure 4) uses a greedy strategy to choose between these two options. Note that the algorithm incorporates the node pullup option directly into Equation 7.

4 Pipelined 2DMTs–Fine Mapping

The development of heuritics for fine grain mapping is similar to that for coarse grain mapping and is omitted for space reasons. Let $MME(W-1, k)$ and $Opt'(root(O), k)$, repectively, be the minimal per-stage memory required by fine grain mapping of an approximate optimal pipelined 2DMT whose dest trie is a FST and the approximate optimal pipelined 2DMT itself whose dest trie is a VST. The time needed to compute $MME(W-1, k)$ and $Opt'(root(O), k)$ is $O(n^2W^3k^2)$ when source tries are VSTs. When source tries are FSTs, the time complexity is $O(nW^3k + nW^2k^2)$ and $O(n^2W^2k^2)$.

```
**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 $Opt'STs(N, i, j).E(l), 0 \leq l < j$.
{
    for ($v = 0; v < j; v++$) $Opt'STs(N, i, j).E(v) = 0$;
    $curMaxE = 0$; // $curMaxE$ represents $\max_{0 \leq l < j}\{Opt'STs(N, i, j).E(l)\}$.
    for (each $M \in D_i(N)$) do {
        $choice1 = max\{(Opt'(M, 1).E(0) + Opt'STs(N, i, j).E(0), curMaxE\}$;
            // node pullup
        for ($v = 0; v < j; v++$) $tmp(v) = Opt'STs(N, i, j).E(v) + Opt'(M, j).E(v)$;
        $choice2 = \max_{0 \leq l < j}\{tmp(l)\}$;
        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 4: Computing $Opt'STs(N, i, j)$ using node pullups

5 Experimental Results

Our algorithms for two-dimensional pipelined 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. We benchmarked our coarse and fine grain two-dimensional multibit trie algorithms against the heuristics of Lu and Sahni [8] for two-dimensional multibit tries that minimize total memory. Lu and Sahni [8] propose 4 heuristics–2DMTa through 2DMTd–to construct two-dimensional multibit tries. Of these, only 2DMTa constructs tries suitable for coarse mapping[4]. Although 2DMTd is the best of the heuristics of [8] (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 [8]. These data sets are derived from 5-dimensional data sets generated using the filter generator of [11]. 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.

[4] 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.

5.1 Coarse Pipeline Mapping

Table 1 gives the reduction in maximum per-stage memory when the tree packing heuristic of [9] is used versus the straightforward mapping. This more sophisticated mapping strategy requires only that if a node N is assigned to stage q of the pipeline, then each descendant of N be assigned to a stage r such that $r > q$. Although the mean reduction (less than 0.3%) is small, the maximum reduction (30%) is significant. All remaining data presented in this section are for the case when the tree packing heuristic of [9] is used.

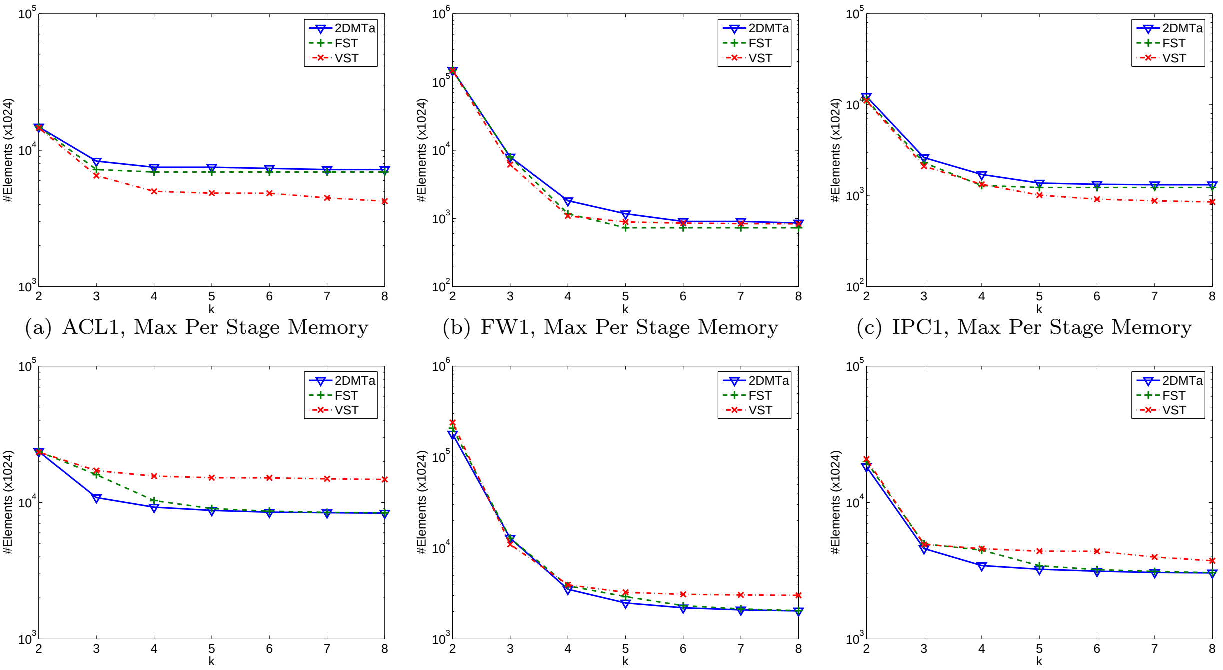
Algorithm	Min	Max	Mean	Standard Deviation
2DMTa	0.00%	30.23%	0.16%	1.83%
FST	0.00%	30.23%	0.27%	2.08%
VST	0.00%	20.00%	0.11%	1.32%

Table 1: 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 $maxSH \in \{3, 4, 5\}$ 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 [9] was employed to reduce the maximum per-stage memory and for VST, the node pullup scheme of Section 3.2 followed by the tree-packing heuristic of [9] were employed. The node pullup scheme reduced the maximum per-stage memory by as much as 80% (the minimum and mean reductions were 0% and 21%, respectively; the standard deviation was 19%) and the tree-packing heuristic reduced this by an additional up to 30% for FSTs and up to 20% on VSTs (as reported in Table 1). Figure 5 shows the maximum per-stage and total memory requirements of the coarse grain mapping resulting from the tries produced by 2DMTa and the FST and VST dest-trie coarse mapping algorithms of Section 3 for the data sets ACL1, FW1 and IPC1 and the case $maxSH = 5$. 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.

On our test data, the maximum per-stage memory of FST normalized by that of 2DMTa is between 0.32 and 1.00; the mean and standard deviation are 0.9 and 0.15, respectively. The normalized numbers for total memory required are between 1.0 and 1.9 with the mean and standard deviation being 1.05 and 0.11. For 10 of our 12 data sets, VST required less maximum per-stage memory than did FST and for the remaining 2, ACL2 and FW1, the reverse is true. For $maxSH = 3$, VST required less maximum per-stage memory on all 12 of our data sets. On our test data, the maximum per-stage memory of VST normalized by that of FST is between 0.24 and 1.29; the mean and standard deviation are 0.83 and 0.2, respectively. The normalized numbers for total memory required are between 0.65 and 1.92 with the mean and standard deviation being 1.21 and 0.26.

Tables 2 and 3 give the normalized memory requirements when VST source tries are used in place of FST source tries. Although the use of VST source tries can reduce memory requirement by 30% or more, the mean reduction is only 3%.



(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 5: Maximum per-stage and total memory required by coarse-grain pipelining, $maxSH = 5$. Source tries are FSTs.

2DMT	Min	Max	Mean	Standard Deviation
2DMTa	0.66	1.00	0.97	0.05
FST	0.70	1.00	0.97	0.04
VST	0.68	1.00	0.97	0.04

Table 2: Maximum per-stage memory required when VST source tries are used normalized by requirement when FST source tries used

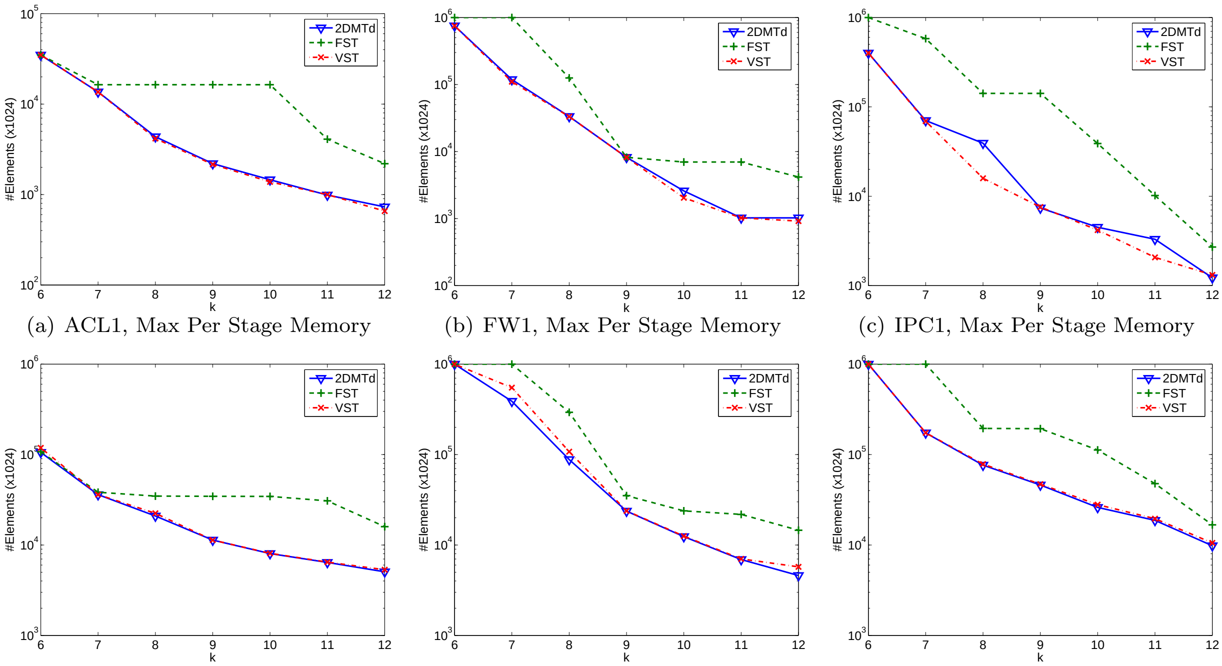
2DMT	Min	Max	Mean	Standard Deviation
2DMTa	0.76	1.00	0.97	0.04
FST	0.64	1.31	0.97	0.05
VST	0.66	1.20	0.97	0.05

Table 3: Total memory required when VST source tries are used normalized by requirement when FST source tries used

5.2 Fine Pipeline Mapping

The tree packing heuristic [9] had a very small impact on our fine-grain pipeline algorithms. The reduction in maximum per-stage memory was at most 3.17%, 0.56% and 4.43%, respectively, for 2DMTd, FST and VST. The mean reductions were 0.15%, 0.01% and 0.22%. Our remaining results are for the case when the tree packing heuristic is used.

Figure 6 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 4. Memory requirements in excess of 10^9 are plotted as 10^9. 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 6: Maximum per-stage and total memory required by fine-grain pipelining. Source tries are FSTs.

For all our data sets, VST outperformed FST on both the maximum per-stage metric as well as the total memory metric. In fact, the maximum per-stage memory for VST normalized by that for FST was between 0.01 (a 99% reduction) and 1.00 (no reduction); the mean and standard deviation were 0.25 and 0.24, respectively. The normalized numbers for total memory were between 0.02 (a 98% reduction) and 1.10 (a 10% increase). The mean and standard deviation were 0.38 and 0.27. FST was always inferior to 2DMTd on both metrics–maximum per-stage memory and total memory. Although VST was generally superior to 2DMTd on the maximum per-stage memory metric, there were times when this was not the case (e.g., IPC1 with $k = 12$). The maximum per-stage memory for VST normalized by that for 2DMTd was between 0.34 and 1.20; the mean and standard deviation were 0.9 and 0.16, respectively. Not surprisingly, 2DMTd was always superior to VST on the total memory metric. The normalized numbers for total memory were between 1.0 and 2.23 and the mean and standard deviation were 1.07 and 0.15.

Tables 4 and 5 show the effect of using VST source tries over FST source tries. The impact on maximum per-stage memory is slightly more than in the case of coarse-gran mapping while the impact on total memory is slightly less.

Figure 7 compares the memory requirements of the tries resulting from the best coarse grain and fine grain

2DMT	Min	Max	Mean	Standard Deviation
2DMTd	0.50	1.07	0.96	0.12
FST	0.52	1.00	0.97	0.08
VST	0.52	1.21	0.97	0.10

Table 4: Maximum per-stage memory when VST source tries are used normalized by requirement when FST source tries are used (fine-grain mapping)

2DMT	Min	Max	Mean	Standard Deviation
2DMTd	0.73	1.00	0.96	0.05
FST	0.73	1.00	0.97	0.05
VST	0.77	1.12	0.97	0.05

Table 5: Total when VST source tries are used normalized by requirement when FST source tries are used (fine-grain mapping)

algorithms (i.e., the VST dest-trie algorithms of Sections 3 and 4). For the case of a coarse grain mapping, we show three cases–$maxSH = 3$ (Coarse-3), 4 and 5. The fine grain mapping is superior to the coarse grain mapping when k is large but not when k is small. For ACL1, for example, the maximum per-stage memory required by the coarse ($maxSH = 5$) and fine mapping is (almost) the same when $k = 8$. However, the cycle time for the 8-stage pipeline is 6 for the coarse mapping and 1 for the fine mapping. So, the fine mapping results in a throughput that is 6 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 391 times the maximum per-stage memory required by a coarse mapping with $maxSH = 5$! When $k = 12$, the fine mapping has a maximum per-stage memory requirement between 4% and 29% that of the coarse mapping with $maxSH = 3$.

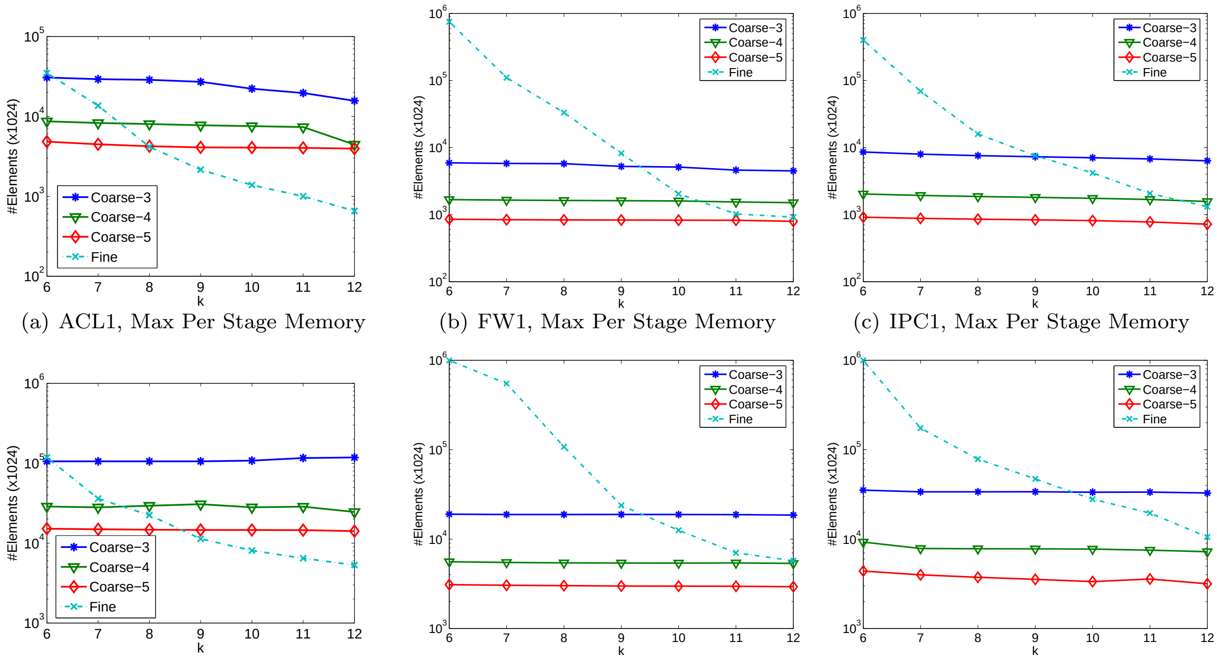
6 Conclusion

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 have a much smaller maximum per-stage memory requirement than do the tries obtained using the 2DMTa algorithm of [8] 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 [8]

References

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

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



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

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

[4] H. Lu and S. Sahni, O(log W) multidimensional packet classification, *IEEE/ACM Transactions on Networking*, to appear.

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

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

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

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

[9] W. Lu and S. Sahni, One-Dimensional Packet Classification Using Pipelined Multibit Tries, *Paper in Review.*

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

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