

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
7 February 2008 (07.02.2008)

PCT

(10) International Publication Number
WO 2008/017045 A2

- (51) International Patent Classification: **Not classified**
- (21) International Application Number: PCT/US2007/075107
- (22) International Filing Date: 2 August 2007 (02.08.2007)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:
60/821,220 2 August 2006 (02.08.2006) US
- (71) Applicant (for all designated States except US): **UNIVERSITY OF FLORIDA RESEARCH FOUNDATION, INC.** [US/US]; 223 Grinter Hall, Gainesville, FL 32611 (US).
- (72) Inventors; and
- (75) Inventors/Applicants (for US only): **LU, Wencheng** [CN/US]; 297 Diamond Village, Apartment 9, Gainesville, FL 32603 (US). **SAHNI, Sartaj** [US/US]; 709 S.w. 80th Blvd., Gainesville, FL 32607 (US).
- (74) Agent: **HINSON, Richard, A.**; Akerman Senterfitt, P.O. Box 3188, West Palm Beach, FL 33402-3188 (US).
- (81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.
- (84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).
- Published:**
— without international search report and to be republished upon receipt of that report

(54) Title: SUCCINCT REPRESENTATION OF STATIC PACKET CLASSIFIERS

(57) Abstract: A computer-implemented method of classifying packets of data transmitted over a data communications network includes providing a memory-storable packet classifier rule defined by a plurality of filters, each filter uniquely corresponding to a one of a plurality of actions to be taken in transmitting the data packets over the data communications network; and classifying a data packet according to the packet classifier rule by generating at least one supernode trie subject to a predetermined constraint for reducing accesses to a memory in which the packet classifier rule is stored. A system including a combination of hardware and software modules is provided for carrying out the method. A computer-readable storage medium is also provided for storing a computer program for carrying out the method.



WO 2008/017045 A2

SUCCINCT REPRESENTATION OF STATIC PACKET CLASSIFIERS

FIELD OF THE INVENTION

[0001] The present invention relates to communication network routers and routing techniques, and more particularly, to the representation of packet classifiers used for routing data packets.

BACKGROUND OF THE INVENTION

[0002] An Internet router classifies incoming packets based on their header fields using a classifier, which is typically implemented as a table of rules. Each classifier rule specifies a pair, (F, A), where F is a filter and A is an action. If an incoming packet matches a filter in the classifier, the associated action specifies what is to be done with this packet. Typical actions include packet forwarding and dropping.

[0003] A d-dimensional filter F is a d-tuple, (F[1], F[2], . . . , F[d]), where F[i] is a range that can specify destination addresses, source addresses, port numbers, protocol types, TCP flags, and other information. A packet is said to match filter F, if the packet's header field values fall in the ranges F[1], . . . , F[d]. Since it is possible for a packet to match more than one of the filters in a classifier, a tie breaker is typically 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. Although 1-dimensional prefix filters are adequate for destination based packet forwarding, higher dimensional filters are generally required for firewall, quality of service, and virtual private network applications, for example.

One-Dimensional Packet Classification

[0004] The filters in a 1-dimensional classifier, as already noted, are assumed to be prefixes of destination addresses. Many of the data structures developed for the representation of a 1-dimensional classifier are based on a binary trie structure. A binary trie is a binary tree structure in which each node has a data field and two children fields. Branching is done based on the bits in the search key. A left child branch is followed at a node at level i (the root is at level 0) if the i th bit of the search key (the leftmost bit of the search key is bit 0) is 0; otherwise a right child branch is followed. Level i nodes store prefixes whose length is i in their data fields. The node

in which a prefix is to be stored is determined by performing a search using that prefix as key. N can correspond to a node in a binary trie. $Q(N)$ can be the bit string defined by the path from the root to N . $Q(N)$ is the prefix that corresponds to N . $Q(N)$ is stored in $N.data$ in case $Q(N)$ is one of the prefixes to be stored in the trie.

[0005] FIG. 1(a) shows a set of 5 prefixes. 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 P_2 is 1. FIG. 1(b) shows the binary trie corresponding to this set of prefixes. Shaded nodes correspond to prefixes in the rule table and each contains the next hop for the associated prefix. The binary trie of FIG. 1(b) differs from a 1-bit trie in that a 1-bit trie stores up to 2 prefixes in a node whereas each node of a binary trie stores at most 1 prefix. Because of this difference in prefix storage strategy, a binary trie may have up to 33 levels when storing IP.v4 prefixes, while the number of levels in a 1-bit trie is at most 32. A binary trie may have up to 129 levels when storing IP.v6 prefixes, while the number of levels in a 1-bit trie is at most 128.

[0006] For any destination address d , the longest matching prefix may be found by following a path beginning at the trie root and dictated by d . The last prefix encountered on this path is the longest prefix that matches d . While this search algorithm is relatively simple, it can result in as many cache misses as the number of levels in the trie. Even for IP.v4, this number, which is at most 33, is too large to classify/forward packets at line speed. Several strategies, such as LC trie, Lulea, tree bitmap, multibit tries, and shape shifting tries, have all been proposed to improve the lookup performance of binary tries. All of these strategies collapse several levels of each subtree of a binary trie into a single node, which is called a supernode, that can be searched with a number of memory accesses that is less than the number of levels collapsed into the supernode. For example, the correct child pointer (as well as its associated prefix) in a multibit trie can be accessed with a single memory access independent of the size of the multibit node. The resulting trie, which is composed of supernodes, is called a supernode trie.

[0007] Data structures have been developed for the compact representation of the supernodes. One such structure is the shape shifting trie (SST) structure proposed by Song, et al. (see "Shape Shifting Tries for Faster IP Route Lookup," Proceedings of 13th IEEE International Conference on Network Protocols, 2005), which in turn draws heavily from the tree bitmap (TBM) scheme of Eatherton, et al. (see "Tree bitmap: hardware/software IP lookups with incremental updates," Computer Communication Review, 34(2): 97-122, 2004) and a technique developed by Jacobson

("Succinct Static Data Structure," Carnegie Mellon University Ph.D Thesis, 1998) for the succinct representation of a binary tree. In TBM, the binary trie is partitioned into subtrees that have at most S levels each. Each partition is then represented as a (TBM) supernode. S is the stride of a TBM supernode. While $S = 8$ is suggested for real-world IPv4 classifiers, here $S = 2$ is used to illustrate the TBM structure.

[0010] FIG. 2(a) shows a partitioning of the binary trie of FIG. 1(b) into 4 subtrees W–Z that have 2 levels each. Although a full binary trie with $S = 2$ levels has 3 nodes, X has only 2 nodes and Y and Z have only one node each. Each partition is represented by a supernode (FIG. 2(b)) that has the following components:

[0011] First, a $(2^S - 1)$ -bit internal bitmap (IBM) that indicates whether each of the up to $2^S - 1$ nodes in the partition contains a prefix. The IBM is constructed by superimposing the partition nodes on a full binary trie that has S levels and traversing the nodes of this full binary trie in level order. For node W, the IBM is 110 indicating that the root and its left child have a prefix and the root's right child is either absent or has no prefix. The IBM for X is 010, which indicates that the left child of the root of X has a prefix and that the right child of the root is either absent or has no prefix (note that the root itself is always present and so a 0 in the leading position of an IBM indicates that the root has no prefix). The IBM's for Y and Z are both 100.

[0012] Second, a 2^S -bit EBM (external bit map) that corresponds to the 2^S child pointers that the leaves of a full S -level binary trie has. The EBM for W is 1011, which indicates that only the right child of the leftmost leaf of the full binary trie is null. The EBMs for X, Y and Z are 0000 indicating that the nodes of X, Y and Z have no children that are not included in X, Y, and Z, respectively. Each child pointer from a node in one partition to a node in another partition becomes a pointer from a supernode to another supernode. To reduce the space required for these inter-supernode pointers, the children supernodes of a supernode are stored sequentially from left to right so that using the location of the first child and the size of a supernode, the location of any child supernode can be computed.

[0013] Third, a child pointer that points to the location where the first child supernode is stored.

[0014] And fourth, a pointer to a list NH of next-hop data for the prefixes in the partition. NH may have up to $2^S - 1$ entries. This list is created by traversing the partition nodes in level order. The NH list for W is nh(P1) and nh(P2), where nh(P1) is the next hop for prefix P1. The NH list

for X is nh(P3). Although the NH pointer is part of the supernode, the NH list is not. The NH list is conveniently represented as an array.

[0015] The NH list (array) of a supernode is stored separate from the supernode itself and is accessed only when the longest matching prefix has been determined, and when it is desired to determine the next hop associated with this prefix. If b bits are needed for a pointer, then a total of $2^{S+1}+2b-1$ bits (plus space for an NH list) are needed for each TBM supernode. Using the IBM, the longest matching prefix in a supernode can be determined; the EBM is used to determine whether it is necessary to move next to the first, second, and so fourth child of the current supernode. If a single memory access is sufficient to retrieve an entire supernode, it is possible to move from one supernode to its child with a single access. The total number of memory accesses to search a supernode trie becomes the number of levels in the supernode trie plus 1 (to access the next hop for the longest matching prefix).

[0016] The SST supernode structure is obtained by partitioning a binary trie into subtrees that have at most K nodes each. K is the stride of an SST supernode. To correctly search an SST, each SST supernode requires a shape bit map (SBM) in addition to an IBM and EBM. The SBM is the succinct representation of a binary tree. It is obtained by replacing every null link in the binary tree being coded by the SBM with an external node, next by placing a 0 in every external node and a 1 in every other node, and finally, by traversing this extended binary tree in level order, listing the bits in the nodes as they are visited by the traversal.

[0017] If the example binary trie of FIG. 1(b) is to be partitioned into binary tries that have at most $K = 3$ nodes each, FIG. 3(a) shows a possible partitioning into the 3 partitions X-Z. X includes nodes a, b and d of FIG. 1(b). Y includes nodes c, e and f. And Z includes node g. The SST representation has 3 (SST) supernodes. The SBMs for the supernodes X-Z, respectively, are 1101000, 1110000, and 100. Note that a binary tree with K internal nodes has exactly $K + 1$ external nodes. So, when the binary trie is partitioned into binary tries that have at most K internal nodes, the SBM is at most $2K+1$ bits long. Since the first bit in an SBM is 1 and the last 2 bits are 0, it is not necessary to store these bits explicitly. Hence, an SBM requires only $2K - 2$ bits of storage. FIG. 3(b) shows the node representation for each partition of FIG. 3(a). The illustrative SBMs exclude the first and last two bits.

[0018] The IBM of an SST supernode is obtained by traversing the partition in level order. When a node is visited, a 1 is output to the IBM if the node has a prefix and a 0 otherwise. The

IBMs for nodes X-Z are, respectively, 110, 011, and 1. Note that the IBM of an SST supernode is at most K bits in length. To obtain the EBM of a supernode, starting with the extended binary tree for the partition, a 1 is placed in each external node that corresponds to a node in the original binary trie and a 0 in every other external node. Next, the external nodes are visited in level order and their bit is output to the EBM. The EBMs for the 3 supernodes are, respectively, 1010, 0000, and 00. Since the number of external nodes for each partition is at most $K + 1$, the size of an EBM is at most $K + 1$ bits.

[0019] As in the case of the TBM structure, child supernodes of an SST supernode are stored sequentially and a pointer to the first child supernode is maintained. The NH list for the supernode is stored in separate memory and a pointer to this list maintained within the supernode. Although the size of an SBM, IBM, and EBM varies with the partition size, an SST supernode is of a fixed size and allocates $2K$ bits to the SBM, K bits to the IBM and $K + 1$ bits to the EBM. Unused bits are filled with 0s. Hence, the size of an SST supernode is $4K + 2b - 1$ bits.

[0020] An $O(m)$ time algorithm, called post-order pruning, has been developed to construct a minimum-node SST, for any given K , from an m -node binary trie. A breadth-first pruning algorithm has also been developed to construct, for any given K , a minimum height SST. The complexity of this algorithm is $O(m^2)$.

[0021] For dense binary tries, TBMs are more space efficient than SSTs. However, for sparse binary tries, SSTs are more space efficient. A hybrid SST (HSST) has been proposed in which dense subtrees of the overall binary trie are partitioned into TBM supernodes and sparse subtrees into SST supernodes. FIG. 4 shows an HSST for the binary trie of FIG. 1(b). For this HSST, $K = S = 2$. The HSST has two SST nodes X and Z, and one TBM node Y.

Two-Dimensional Packet Classification

[0022] The filters are assumed to be of the form (D, E) , where D is a destination address prefix and E is a source address prefix. A 2-dimensional classifier may be represented as a 2-dimensional binary trie (2DBT), which is a one-dimensional binary trie (called the top-level trie) in which the data field of each node is a pointer to a (possibly empty) binary trie (called the lower-level trie). So, a 2DBT has 1 top-level trie and potentially many lower-level tries.

[0023] FIG. 5 illustrates an exemplary 5-rule two-dimensional classifier. For each rule, the filter is defined by the destination (Dest) and source (Source) prefixes. So, for example, $F2 = (0*, 1*)$ matches all packets whose destination address begins with 0 and whose source address begins with 1. When a packet is matched by two or more filters, the matching rule with least cost is used. The classifier of FIG. 5 may be represented as a 2DBT 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). N is a node in the destination trie. If no dest prefix equals $Q(N)$, then $N.data$ points to an empty lower-level trie. If there is a dest prefix D that equals $Q(N)$, then $N.data$ points to a binary trie for all source prefixes, E , such that (D, E) is a filter. In the context of destination-source filters, the lower-level tries are called source trees. FIG. 6 gives the 2DBT for the filters of FIG 5.

[0024] Two-dimensional one-bit tries, a close relative of 2DBTs, have been proposed for destination-source prefix filters. The proposed two-dimensional trie structure takes $O(nW)$ memory, where n is the number of filters in the classifier and W is the length of the longest prefix. Using this structure, a packet may be classified with $O(W^2)$ memory accesses. The basic two-dimensional one-bit trie may be improved upon by using pre-computation and switch pointers. The improved version classifies a packet making only $O(W)$ memory accesses. Extensions to higher-dimensional one-bit tries have also been proposed, which may be used with d -dimensional, $d > 2$, filters. Two-dimensional one-bit tries with buckets can also be used for d -dimensional, $d > 2$, classifiers. Basically, the destination and source fields of the filters are used to construct a two-dimensional 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. This scheme is expected to work well in practice because the bucket size tends to be small. Also, switch pointers may not be used in conjunction with the bucketing scheme. Fast polynomial-time algorithms have been developed to construct space-optimal constrained 2DMTs (two-dimensional multibit tries). The constructed 2DMTs may be searched with at most k memory accesses, where k is a design parameter. The space-optimal constrained 2DMTs may be used for d -dimensional filters, $d > 2$, using the bucketing strategy. For the case $d = 2$, switch pointers may be employed to get multibit tries that require less memory than required by space-optimal constrained 2DMTs and that permit packet classification with at most k memory

accesses. A fast heuristic has also been developed to construct good multibit tries with switch pointers. Experiments indicate that, given the same memory budget, space-optimal constrained 2DMT structures perform packet classification using 1/4 to 1/3 as many memory accesses as required by the two-dimensional one-bit tries.

SUMMARY OF THE INVENTION

The present invention provides a memory-efficient classifier representation that can be efficiently searched with a reduced number of memory accesses and a reduction in the total amount of computer memory required. One aspect of the invention is a procedure, based on a fast algorithm for constructing minimum-height SSTs, that reduces the complexity of the construction from $O(m^2)$ to $O(m)$, where m is the number of nodes in the input binary trie. Other aspects of the invention are dynamic programming procedures for the construction of space-optimal HSSTs and good 2DHSSTs and 2DHSSTPCs (constrained 2DHSSTs with prefix inheritance).

BRIEF DESCRIPTION OF THE DRAWINGS

[0025] There are shown in the drawings, embodiments which are presently preferred. It is expressly noted, however, that the invention is not limited to the precise arrangements and instrumentalities shown.

[0026] FIG. 1(a) is a schematic representation of an exemplary set of 5 prefixes and the corresponding binary trie;

[0027] FIG. 1(b) is a schematic representation of a binary trie corresponding to the prefixes of FIG. 1(a);

[0028] FIG. 2 is a schematic representation of a partitioning of the binary trie of FIG. 1(b) using a TBM scheme;

[0029] FIG. 3 is a schematic representation of a partitioning of the binary trie of FIG. 1(b) using an SST scheme;

[0030] FIG. 4 is a schematic representation of a partitioning of the binary trie of FIG. 1(b) using a HSST scheme;

[0031] FIG. 5 is a schematic representation of exemplary five dest-source filters;

[0032] FIG. 6 is a schematic representation of a 2DBT for FIG. 5;

[0033] FIG. 7 is schematic view of a data communications network in which a system, according to a particular embodiment of the invention, is deployed;

[0034] FIG. 8 schematically illustrates the construction of a two-dimensional supernode trie (2DHSST) for Fig. 6, according to a particular embodiment of the invention;

[0035] FIG. 9 schematically illustrates the construction of a 2DHSST with prefix inheritance (2DHSSTP) for FIG. 8, according to a particular embodiment of the invention;

[0036] FIG. 10 schematically illustrates the construction of four leaf supernode formats, according to a particular embodiment of the invention;

[0037] FIG. 11 is a graphical representation of the number of memory accesses required for a lookup in IP4v tables using each of the algorithmic procedure of the invention;

[0038] FIG. 12 is a graphical representation of the number of memory accesses shown in FIG. 11 in column chart;

[0039] FIGS. 13(a) and (b) are tables of access and memory statistics, respectively, for IP4v data normalized by EBO data;

[0040] FIG. 14 is a graphical representation of the total memory required by IP4v tables;

[0041] FIGS. 15(a) and (b) are graphical representations of the required total memory as shown in FIGS. 14(a) and (b) in column charts, according to a particular embodiment of the invention;

[0042] FIG. 16 is a table of amounts of memory required by BARTs;

[0043] FIG. 17 is a table of the number of memory accesses required for a lookup in IP4v tables by each of the six algorithmic procedures implemented according to a particular embodiment of the invention;

[0044] FIG. 18 is a graphical representation of the number of memory accesses shown in FIG. 17;

[0045] FIGS. 19(a) and (b) are tables of access and memory statistics, respectively, for IP6v data normalized by EBO data;

[0046] FIG. 20 is a table the total memory required by IP6v tables;

[0047] FIG. 21 is a graphical representation of the required total memory as shown in FIG. 20;

[0048] FIG. 22 is a table of the total memory and number accesses required by IP4v tables after optimization according to a particular embodiment of the invention;

[0049] FIG. 23 is a table of the total memory and number of memory accesses required by 2DHSSTs and 2DHSSTPCs of the invention;

[0050] FIG. 24 is a graphical representation of the total memory required by 2DHSSTs and 2DHSSTPCs of the invention;

[0051] FIG. 25 is a graphical representation of the number of memory accesses for a 2D lookup performed according to a particular embodiment of the invention;

[0052] FIG. 26 is a table of the total memory required by 2DHSSTPCs, 2DMTDs, and Grid-of-Tries;

[0053] FIGS. 27(a) and (b) are tables of total memory (KBytes) and number of memory accesses required, respectively, by 2DHSSTPCs and extended 2DHSSTPCs of the invention;

[0054] FIG. 28 is a table of total memory (KBytes), bits/rule, and number of memory accesses required by HyperCuts on 5-dimensional data sets; and

[0055] FIGS. 29(a) and (b) are graphical representations of total memory (KBytes) and number of memory accesses required by HyperCuts and extended 2DHSSTPCs, according to a particular embodiment of the invention.

[0056] FIG. 30 is a flowchart of exemplary steps of a method for classifying packets of data transmitted over a data communications network, according to a particular embodiment of the invention.

DETAILED DESCRIPTION

[0057] Figure 7 schematically illustrates a data communication network 20, which is an exemplary environment in which a system 100, according to one embodiment, can be deployed. The data communication network 20 illustratively comprises a plurality of hosts 22a, 22b, only two of which are shown, a plurality of nodes 24a-h, 26a-e to which the hosts are connected, and two additional exemplary networks 28a, 28b connected to the plurality of nodes. Those skilled in the art will readily appreciate that the data communication network 20 can be the Internet, a local area network (LAN), for example, or any other network formed by two or more hosts whose application-layer programs communicate with one another by transmission of packetized data routed through the various nodes. The nodes 24a-h, 26a-e can be routers, hubs, bridges, or any other general purpose computer or application-specific processing device able to store and forward data packets based on stored routing information. The plurality of nodes 24a-h, 26a-e classify incoming data packets into flows based on information contained in the headers of the

packets and on information each node stores in its router table. Each entry in the table is a filter-action pair.

[0058] The system 100, more particularly, is a system for classifying the packets of data transmitted over the data communications network 20. Though shown as a distinct element for ease of presentation, the system 100 can be implemented in one or more nodes 24a-h, 26a-e of the data communications network 20. As shown, the system 100 is communicatively linked to the various nodes 24a-h, 26a-e of the data communications network 20.

[0059] The system 100 comprises a combination of hardware and software modules for classifying packets of data transmitted over the data communications network 20. Illustratively, the system includes a memory 102. Within the memory 102, at least one packet classifier rule defined by a plurality of filters is stored as computer-readable code. Each filter uniquely corresponds to a one of a plurality of actions to be taken in transmitting the data packets over the data communications network 20.

[0060] Additionally, the system 100 includes a processor 104 comprising logic circuitry (not explicitly shown) that responds to and processes computer-readable instructions or code. Specifically, the processor 104 is configured to access the packet classifier rule in the memory 102 and to classify a data packet based upon the packet classifier rule. As described more particularly below, the data packet is classified by the processor 104 generating at least one supernode trie subject to a predetermined constraint for reducing accesses to the memory. The supernode trie is explicitly defined in context of the algorithmic processes described below.

[0061] According to one embodiment, the classifier rule comprises a 1-dimensional node, space-optimal supernode and wherein the predefined constraint minimizes memory accesses performed by the processor in classifying the data packet. According to another embodiment, the classifier rule comprises a multi-dimensional node, space-optimal supernode, wherein the predefined constraint minimizes memory accesses performed by the processor in classifying the data packet. Thus, the processor 104, according to this embodiment, can be configured to generate a supernode that corresponds to a predefined constraint that minimizes storage of the at least one classifier rule within the memory.

[0062] The various embodiments of the invention include algorithm-based procedures, which can be implemented in computer-readable code, for succinctly representing 1-dimensional tries

and 2-dimensional tries. The algorithms are described herein, along with rigorous proofs demonstrating the logical basis of each.

Minimum-Height SSTs

[0063] The invention encompasses algorithms for the succinct representation packet classifiers. More particularly, the embodiments of the invention include algorithms for the succinct representations of 1-dimensional tries and 2-dimensional tries.

[0064] The breadth-first pruning algorithm constructs, for any given K and binary trie T , a minimum height SST. The complexity of this algorithm is $O(m^2)$, where m is the number of nodes in T . The present invention provides an $O(m)$ algorithm for this task. This algorithm, referred to herein as minHtSST, performs a postorder traversal of T . When a node x of T is visited during this traversal, one or both of the currently remaining subtrees of x and, at times, even the entire remaining subtree rooted at x may be pruned off to form a node of the SST being constructed.

[0065] When minHtSST visits a node x of T , some (or all) of the descendants of x in T have been pruned by earlier node visits. The pruned descendants of x have been mapped into supernodes that form one or more SSTs. These SSTs are referred to as the SSTs that hang from x . Some of these SSTs that hang from x were created during visits of nodes in the left subtree of x . These SSTs are called the left hanging SSTs; the remaining SSTs are the right hanging SSTs of x . The following notation is used: $x.\text{leftChild}$ ($x.\text{rightChild}$) is the left (right) child of x in T ; $x.\text{st}$ is the set of nodes in the subtree of T rooted at x ; $x.\text{rn}$ (remaining nodes) is the subset of $x.\text{st}$ that have not been pruned off at the time x is visited; $x.\text{size}$ is the number of nodes in $x.\text{rn}$; $x.\text{SST s}$ is the set of SSTs that hang from x at the time x is visited; $x.\text{leftSST s}$ ($x.\text{rightSST s}$) is the subset of $x.\text{SST s}$ that are left (right) hanging SSTs. $x.\text{lht} = -1$ (left height) if $x.\text{leftSST s}$ is empty. Otherwise, $x.\text{lht}$ is the maximum height of an SST in $x.\text{leftSST s}$ (the height of an SST is 1 less than the number of levels in the tree). $x.\text{rht}$ is the corresponding quantity for the $x.\text{rightSST s}$ and $x.\text{ht} = \max\{x.\text{lht}, x.\text{rht}\}$.

[0066] The function $\text{prune}(y)$ prunes T at the node y by removing all nodes in $y.\text{rn}$. The nodes in $y.\text{rn}$ are used to create a supernode whose subtrees are $y.\text{SST s}$. When y is NULL, $\text{prune}(y)$ is a NULL operation. The following is the visit function employed by the postorder traversal algorithm minHtSST:

```

Case 1:  $[x.lht == x.rht]$ 
  if ( $x.size > K$ ) {prune( $x.leftChild$ ); prune( $x.rightChild$ );}
  else if ( $x.size == K$ ) prune( $x$ );
  return;

Case 2:  $[x.lht < x.rht]$ 
  prune( $x.leftChild$ );
  update  $x.size$ ;
  if ( $x.size == K$ ) prune( $x$ );
  return;

Case 3:  $[x.lht > x.rht]$ 
  Symmetric to Case 2.

```

[0067] x is the node of T being visited. This visit function has 3 mutually exclusive cases. Exactly one of these is executed during a visit.

[0068] As demonstrated herein, if T is traversed in postorder using the visit function, then $x.leftChild.size < K$ and $x.rightChild.size < K$ when x is visited. Less evident is the fact that when x is visited, every node y that is in the left (right) subtree of x and in $x.rn$ has $y.ht = x.lht$ ($x.rht$).

[0069] Lemma 1: When x is visited, every node y that is in the left (right) subtree of x and in $x.rn$ has $y.ht = x.lht$ ($x.rht$).

[0070] Proof: u is the nearest left descendent (i.e., descendent in the left subtree) of x that is in $x.rn$ and has $u.ht \neq x.lht$. If there is no such u , then $y.ht = x.lht$ for every left descendent y of x that is in $x.rn$. So, assume there is such a u . Clearly, $u.ht < x.lht$. so u cannot be the left child of x as otherwise $x.leftSST.s = u.SST.s$ and so $x.lht = u.ht$. Therefore, u has a parent v that is in $x.rn$ (v also is a left descendent of x) and $u.ht < v.ht = x.lht$. Without loss of generality, assume that u is the left child of v . Thus, $v.lht = u.ht < v.ht = v.rht$. During the postorder traversal of T , node v is visited before node x . When v was visited, v 's left subtree (including node u) should have been pruned from T (Case 2 of the visit function) and so u cannot be in $x.rn$, which is a contradiction.

[0071] The proof for nodes that are in the right subtree of x and in $x.rn$ is similar.

[0072] Theorem 1: For every binary trie T and integer $K > 0$, the postorder traversal algorithm *minHtSST* constructs an SST that has minimum height.

[0073] Proof: The theorem can be established by showing that if there is a minimum height SST for T that includes all the supernodes constructed up to (but not including) the time a node x is visited, then there is a minimum height SST for T that includes all the supernodes constructed up

to and including the visit of x . Since the antecedent of this statement is true when the first node of T is visited, the theorem follows by induction.

[0074] Consider the visit of any node x of T . U is an SST that includes all supernodes constructed by the algorithm up to this time. If, during the visit of x , no new supernodes are constructed, then there is nothing to proven. Therefore, assume that one or more supernodes are constructed. P denotes the (super)node of U that includes x .

[0075] Suppose the new supernodes are constructed in Case 1 of the visit function. So, $x.lht = x.rht$ and $x.size \geq K$. If $x.size > K$, P cannot include all nodes of $x.rn$. Thus, some of the nodes of $x.rn$ are contained in descendents of P . From Lemma 1 above, it can be obtained $y.ht = x.lht = x.rht = x.ht$ for every $y \in x.rn$ other than $y = x$. Hence, the height of the subSST of U rooted at P is at least $x.ht + 2$. Now, delete the nodes of $x.rn$ (other than x) from the supernodes of U , delete from U any supernodes that become empty, and add to U the two supernodes constructed by $\text{prune}(x.leftChild)$ and $\text{prune}(x.rightChild)$. An SST U' for T is obtained whose height is no more than that of U and which contains all the supernodes constructed up to and including the visit of x . If $x.size = K$, P may include all nodes of $x.rn$. In this case, U has all the supernodes constructed by the algorithm up to and including the visit of x . If P does not include all nodes of $x.rn$, some nodes of $x.rn$ must be in descendent nodes of P and so, as before, the height of the subSST rooted at P is at least $x.ht + 2$. Now, delete the nodes of $x.rn$ from the supernodes of U , delete from U any supernodes that become empty, and add to U the supernode constructed by $\text{prune}(x)$. An SST U' for T is obtained whose height is no more than that of U and which contains all the supernodes constructed up to and including the visit of x .

[0076] If the new supernode is constructed in Case 2, the height of the subSST of T rooted at P is at least $x.rht + 1$. Delete the left descendents of x that are in $x.rn$ from the supernodes of U , delete from U any supernodes that become empty, and add to U the supernode constructed by $\text{prune}(x.leftChild)$. An SST U' for T is obtained whose height is no more than that of U . Now, if $x.size = K$, do the transformation given above in Case 1 ($x.size = K$) and obtain an SST whose height is no more than that of U and which contains all the supernodes constructed up to and including the visit of x .

[0077] The proof for Case 3 is similar to that for Case 2.

[0078] Since the visit function can be implemented to run in $O(1)$ time, the complexity of the postorder traversal function minHtSST is $O(m)$ where m is the number of nodes in the binary trie T . Note that the number of nodes in the binary trie for n prefixes whose length is at most W is $O(nW)$. So, in terms of n and W , the complexity of minHtSST is $O(nW)$.

Space-Optimal HSSTs

[0079] Let $\text{minSpHSST}(T, H)$ be a minimum space HSST for the binary trie T under the restrictions that the stride of the TBM nodes is S and that of the SST nodes is K and the height of the HSST is at most H . Assume that S and K are such that the size of a TBM supernode is the same as that of an SST supernode. Although it may not be possible to choose S and K so that the number of bits needed by a TBM supernode is exactly equal to that needed by an SST supernode, in practice, node size is chosen to match the bandwidth of the memory. This means that a few bits are wasted in every supernode, if necessary, to ensure a supernode size equal to the memory bandwidth. So, in practice, with the wasted memory factored in, the size of a TBM supernode equals that of an SST supernode. Hence, minimizing the space required by an HSST is equivalent to minimizing the number of supernodes in the HSST. Therefore, the number of supernodes in an HSST is used as a measure of its space requirement.

[0080] Let $ST(N)$ denote the subtree of T that is rooted at node N . So, $T = ST(\text{root}(T))$. Let $\text{opt}(N, h)$ be the number of supernodes in $\text{minSpHSST}(ST(N), h)$. $\text{opt}(\text{root}(T), H)$ is the number of supernodes in $\text{minSpHSST}(T, H)$. The present invention provides a dynamic programming recurrence for $\text{opt}(N, h)$. This recurrence may be solved to determine $\text{opt}(\text{root}(T), H)$. A simple extension to the recurrence can be used to actually compute $\text{minSpHSST}(T, H)$.

[0081] Let $\text{opt}(N, h, k)$ be the number of supernodes in a space-optimal HSST for $ST(N)$ under the restrictions: (a) the root of the HSST is an SST supernode for exactly k , $0 < k \leq K$, nodes of the binary trie $ST(N)$ (k is the utilization of the SST node) and (b) the height of the HSST is at most h . Let $D_t(N)$ be the descendants (in T) of N that are at level t of $ST(N)$.

[0082] There are two possibilities for the root of $\text{minHSST}(ST(N), h)$, $h \geq 0$: the root is a TBM supernode or the root is an SST supernode. In the former case,

$$\text{opt}(N, h) = 1 + \sum_{R \in D_1(N)} \text{opt}(R, h-1) \quad (1)$$

and in the latter case,

$$opt(N, h) = \min_{0 < k \leq K} \{opt(N, h, k)\} \quad (2)$$

[0083] Combining these two cases together, it can be obtained

$$opt(N, h) = \min \left\{ 1 + \sum_{R \in D_N(N)} opt(R, h-1), \min_{0 < k \leq K} \{opt(N, h, k)\} \right\} \quad (3)$$

[0084] To simplify the recurrence for $opt(N, h, k)$, the function $f(N, h, k)$ is used, which gives the number of supernodes in the space-optimal HSST for the binary trie composed of ST (N) and the parent of N (assuming that N is not the root of T) under the restrictions: (a) the root of the HSST is an SST supernode whose utilization is $k + 1$ and (b) the height of the HSST is at most h . Note that when $k = 0$, the root of this HSST contains only the parent of N. So, $f(N, h, 0) = 1 + opt(N, h-1)$. When $k > 0$, the root represents a partition that includes the parent of N plus k nodes of ST (N). So, $f(N, h, k) = opt(N, h, k)$. To obtain the recurrence for $opt(N, h, k)$, $h > 0$ and $k > 0$, consider three cases, namely N has 0, 1, and 2 children, are considered.

[0085] When N has no child,

$$opt(N, h, k) = 1 \quad (4)$$

[0086] When N has only one child a,

$$opt(N, h, k) = f(a, h, k-1) \quad (5)$$

[0087] When N has two children a and b,

$$opt(N, h, k) = \min_{0 \leq j \leq k} \{f(a, h, j) + f(b, h, k-j-1) - 1\} \quad (6)$$

[0088] Finally, for $h < 0$,

$$opt(N, h, k) = opt(N, h) = \infty \quad (7)$$

and for $k \leq 0$,

$$opt(N, h, k) = \infty \quad (8)$$

as it is not possible to represent ST (N) by an HSST whose height is less than 0 or by an HSST whose root is an SST node with utilization ≤ 0 .

[0089] Using Equation 3, each $opt(*, *)$ value can be computed in $O(K)$ time, since $|D_S(N)| \leq 2^S \approx 2K$. Also, each $opt(*, *, *)$ value can be computed in $O(K)$ time using Equations 4-8. There are $O(mH)$ $opt(*, *)$ and $O(mHK)$ $opt(*, *, *)$ values to compute. Hence, the time complexity is $O(mHK + mHK^2) = O(mHK^2) = O(nWHK^2)$, where n is the number of filters and W is the length of the longest prefix.

Space-Optimal 2DHSSTs

[0090] Let T be a 2DBT. Assume that the source tries of T have been modified so that the last prefix encountered on each search path is the least-cost prefix for that search path. This modification is accomplished by examining each source-trie node N that contains a prefix and replacing the contained prefix with the least-cost prefix on the path from the root to N . A 2DHSST may be constructed from T by partitioning the top-level binary trie (i.e., the dest trie) of T and each lower-level binary trie into a mix of TBM and SST supernodes. Supernodes that cover the top-level binary trie use their NH (next hop) lists to store the root supernodes for the lower-level HSSTs that represent lower-level tries of T .

[0091] FIG. 8 shows a possible 2DHSST for the 2DBT of FIG. 6. The supernode strides used are $K = S = 2$. A 2DHSST may be searched for the least-cost filter that matches any given pair of destination and source addresses (da , sa) by following the search path for da in the destination HSST of the 2DHSST. All source tries encountered on this path are searched for sa . The least-cost filter on these source-trie search paths that matches sa is returned. Suppose finding the least-cost filter that matches (000,111), the search path for 000 first starts from the root (ab) of the 2DHSST of FIG. 8 and then goes to the left child (dg). In the 2DHSST root, the search goes through nodes a and b of the dest binary trie and in the supernode dg through nodes d and g of T . Three of the encountered nodes (a , b , and g) have a hanging source trie. The corresponding source HSSTs are searched for 111 and F2 is returned as the least-cost matching filter.

[0092] To determine the number of memory accesses required by a search of a 2DHSST, assume sufficient memory bandwidth that an entire supernode (this includes the IBM, EBM, child and NH pointers) may be accessed with a single memory reference. To access a component of the NH array, an additional memory access is required. For each supernode on the search path for da , one memory access is made to get the supernode's fields (e.g., IBM, EBM, child and NH pointers). In addition, for each supernode on this path, it is necessary to examine some number of hanging source HSSTs. For each source HSST examined, first a component of the dest-trie supernode's NH array is accessed to get the root of the hanging source HSST. Then this hanging source HSST is searched by accessing the remaining nodes on the search path (as determined by the source address) for this HSST. Finally, the NH component corresponding to the last node on this search path is accessed. So, in the case of the above example, 2 memory accesses are made

to fetch the 2 supernodes on the dest HSST path. In addition, 3 source HSSTs are searched. Each requires an access to its root supernode plus an NH component in each source HSST. The total number of memory accesses is $2 + 2 * 3 = 8$.

[0093] Let $MNMA(X)$ be the maximum number of memory accesses (MNMA) required to search a source HSST X . For a source HSST, the MNMA includes the access to NH component of the last node on the search path. So, $MNMA(X)$ is one more than the number of levels in X . Let U be a 2DHSST for T with strides S and K . Let P be any root to leaf path in the top level HSST of U . Let the sum of the MNMAs for the lower-level HSSTs on the path P be $H(P)$. Let $nodes(P)$ be the number of supernodes on the path P . Define 2DHSST (h) to be the subset of the possible 2DHSSTs for T for which

$$\max_P \{H(P) + nodes(P)\} \leq h \quad (9)$$

[0094] Note that every U , $U \in 2DHSST(h)$, can be searched with at most h memory accesses per lookup. Note also that some 2DHSSTs that have a path P for which $H(P) + nodes(P) = h$ can be searched with fewer memory accesses than h as there may be no (da, sa) that causes a search to take the longest path through every source HSST on paths P for which $H(P) + nodes(P) = h$.

[0095] Consider the construction of a space-optimal 2DHSST V such that $V \in 2DHSST(H)$, such a V being referred to as a space-optimal 2DHSST (h). Let N be a node in T 's top-level trie, and let 2DBT (N) be the 2-dimensional binary trie rooted at N . Let $opt1(N, h)$ be the size (i.e., number of supernodes) of the space-optimal 2DHSST (h) for 2DBT (N). $opt1(root(T), H)$ gives the size of a space-optimal 2DHSST (H) for T . Let $g(N, q, h)$ be the size (excluding the root) of a space-optimal 2DHSST (h) for 2DBT (N) under the constraint that the root of the 2DHSST is a TBM supernode whose stride is q . So, $g(N, S, h) + 1$ gives the size of a space-optimal 2DHSST (h) for 2DBT (N) under the constraint that the root of the 2DHSST is a TBM supernode whose stride is S . It can be seen that, for $q > 0$,

$$g(N, q, h) = \min_{m(N) \leq i \leq h} \{g(LC(N), q-1, h-i) + g(RC(N), q-1, h-i) + s(N, i)\} \quad (10)$$

where $m(N)$ is the minimum possible value of MNMA for the source trie (if any) that hangs from the node N (in case there is no source trie hanging from N , $m(N) = 0$), $g(N, 0, h) = opt1(N, h - 1)$, $g(null, t, h) = 0$, and $LC(N)$ and $RC(N)$ respectively, are the left and right children (in T) of N . $s(N, i)$ is the size of the space-optimal HSST for the source trie that hangs off from N under

the constraint that the HSST has an MNMA of at most i . $s(N, i)$ is 0 if N has no hanging source trie.

[0096] Let $opt1(N, h, k)$ be the size of a space-optimal 2DHSST (h) for 2DBT (N) under the constraint that the root of the 2DHSST is an SST supernode whose utilization is k . It is easy to see that

$$opt1(N, h) = \min\{g(N, S, h) + 1, \min_{0 < k \leq K} \{opt1(N, h, k)\}\} \quad (11)$$

[0097] Suppose that $k > 0$ and $h > 0$. If N has no child,

$$opt1(N, h, k) = 1 + s(N, h - 1) \quad (12)$$

[0098] When N has only one child a ,

$$opt1(N, h, k) = \min_{m(N) \leq i \leq h} \{f(a, h - i, k - 1) + s(N, i)\} \quad (13)$$

where $f(N, h, k)$ is the size of a space-optimal 2DHSST (h) for 2DBT (N) plus the parent (in T) of N (but excluding the lower-level source trie (if any) that hangs from N) under the constraint that the root of the 2DHSST is an SST supernode whose utilization is $k+1$. For example, when $k = 0$, the root of the constrained 2DHSST has a utilization 1 and contains only the parent of N ; the remaining supernodes of the 2DHSST represent 2DBT (N). Thus $f(N, h, k) = opt1(N, h, k)$ when $k > 0$ and $1 + opt1(N, h - 1, 0)$ when $k = 0$.

[0099] When N has two children a and b ,

$$opt1(N, h, k) = \min_{m(N) \leq i \leq h} \left\{ \min_{0 \leq j \leq k} \{f(a, h - i, j) + f(b, h - i, k - j - 1) - 1\} + s(N, i) \right\} \quad (14)$$

[00100] For $h \leq 0$

$$opt1(N, h, *) = \infty \quad (15)$$

[00101] When there are n filters and the length of the longest prefix is W , the number of nodes in the dest trie of T is $O(nW)$ and the number of source tries in T is $O(n)$. The time to compute all $s(N, h)$ values using the algorithm described in Section "Space-Optimal 2DHSSTs" to compute opt is $O(n^2WHK^2)$ time. Using Equation 10 and previously computed g values, $O(H)$ time is needed to compute each $g(*, *, *)$ value. Using Equation 11, each $opt1(*, *)$ value may be computed in $O(K)$ time. Using Equations 12-15, each $opt1(*, *, *)$ value can be computed in $O(KH)$ time. Since there are $O(nWH)$ $opt1(*, *)$, $O(nWHK)$ $opt1(*, *, *)$, and

$O(nWSH) g(*, *, *)$ values to compute, the time to determine $\text{opt1}(\text{root}(T), H)$ is $O(n^2WHK^2 + nWHK + nW H^2 K^2 + nWS H^2) = O(n^2WHK^2)$ (as, in typical applications, $n > H$).

2DHSSTs With Prefix Inheritance (2DHSSTP)

[0100] Let T be the 2DBT of FIG. 6. Consider the dest-trie supernode ab of FIG. 8. This supernode represents the subtree of T that is comprised of the binary nodes a and b . A search in this subtree has three exit points: left child of b , right child of b , and right child of a . For the first two exit points, the source tries that hang off of a and b are searched whereas for the third exit point, only the source trie that hangs off of a is searched. In other words, the first two exit points use the source tries that hang off of a and b while the third exit point uses only the source trie that hangs off of a . If the source trie that hangs off of b is augmented with the prefixes in the source trie that hangs off of a , then when the first two exit points are used, only the augmented source trie that hangs off of b need be searched.

[0101] In prefix inheritance, each non-empty source trie in a partition is augmented with the prefixes in all source tries that hang off of ancestors in the partition. When this augmentation results in duplicate prefixes, the least-cost prefix in each set of duplicates is retained. The resulting augmented source tries are called exit tries. In a 2DHSST with prefix inheritance (2DHSSTP), prefix inheritance is done in each supernode. FIG. 9 gives the 2DHSSTP for the 2DHSST of FIG. 8.

[0102] Notice that to search a 2DHSSTP, at most one exit trie for each dest-trie supernode encountered needs to be searched: the last exit trie encountered in the search of the partition represented by that dest-trie supernode. So, when searching for $(da, sa) = (000, 111)$, the exit tries that hang off of b and g are searched for 111. The number of memory accesses is 2 (for the two supernodes ab and dg) + 2 (to access the supernode in each of the two source tries searched) + 2 (to access the NH arrays for the source trie supernodes) = 6. The same search using the 2DHSST of FIG. 8 will search three source tries (those hanging off of a , b , and g) for a total cost of 8 memory accesses.

[0103] A node N in a dest-trie partition is a dominating node if there is an exit trie on every path from N to an exit point of the partition. Notice that if N has two children, both of which are dominating nodes, then the exit trie (if any) in N is never searched. Hence, there is no need to store this exit trie.

[0104] A good algorithm to construct a space-optimal constrained 2DHSSTP for any 2DBT T has been developed by the inventors of the instant application. Note that the 2DHSSTP for T is comprised of supernodes for the dest-trie of T plus supernodes for the exit tries.

[0105] Let 2DHSSTPC(h) be a 2DHSSTP that is constrained so that (a) it can be searched with at most h memory accesses and (b) the HSST for each exit trie is a minimum height HSST for that exit trie. The experimental studies suggest that the space required by an HSST is somewhat insensitive to the height constraint placed on the HSST. So, the space required by a space-optimal 2DHSSTPC(h) is expected to be close to that required by a space-optimal 2DHSSTP(h).

[0106] Let N be a node in the dest-trie of the 2DBT T and let $\text{opt2}(N, h)$ be the size of a space-optimal 2DHSSTPC(h) for the subtree, $ST(N)$, of T rooted at N . The supernode strides are K and S . Notice that $\text{opt2}(\text{root}(T), H)$ gives the size of a space-optimal 2DHSSTPC(H) for T . The development of a dynamic programming recurrence for opt2 follows the pattern used for the earlier dynamic programming recurrences. Suppose that the root of the space-optimal 2DHSSTPC(N) is a TBM supernode. Then,

$$\text{opt2}(N, h) = 1 + ss(N) + \sum_{R \in Ds(N)} \text{opt2}(R, h - 1 - h(R)) \quad (16)$$

where $ss(N)$ is the sum of the sizes of the minimum height HSSTs for the exit tries of the root TBM supernode and $h(R)$ is the MNMA for the last exit trie (if any) of the root that is on the path to R ; if there is no exit trie on this path, then $h(R) = 0$.

[0107] The only other possibility for the root of the 2DHSSTPC(h) is that it is an SST node whose occupancy is k for some k in the range $[1, K]$. Let 2DHSSTPC(N, h, k, p) be a 2DHSSTPC(h) for $ST(N)$ under the constraints (a) the root of the 2DHSSTPC is an SST node whose utilization is k and (b) for the root, prefix inheritance is not limited to the partition of T represented by the root of the 2DHSSTPC; rather prefix inheritance extends up to the p nearest ancestors of N in T . Let $\text{opt2}(N, h, k, p)$ be the size of a space-optimal 2DHSSTPC(N, h, k, p). It can be seen that:

$$\text{opt2}(N, h) = \min_{0 < k \leq K} \{ \text{opt2}(N, h, k, 0) \} \quad (17)$$

[0108] To facilitate the computation of $\text{opt2}(N, h, k, p)$, three new functions: $s(N, p)$, $h(N, p)$ and $x(N, h, k, p)$ are introduced. If N has a non-empty source trie, then $s(N, p)$ is the size of a space-optimal minimum-height HSST for the union of the source tries that hang off of N and its p

nearest ancestors in T and $h(N, p)$ is the MNMA for this HSST. Otherwise, $s(N, p) = h(N, p) = 0$. The $s(N, p)$ values are computed prior to this postorder traversal using the algorithm of Section "Space-Optimal 2DHSSTs." The $h(N, p)$ values are computed easily during the computation of the $s(N, p)$ values.

[0109] $x(N, h, k, p)$ is the size of a space-optimal 2DHSSTPC(N, h, k, p) under the added constraint that the root of the 2DHSSTPC(N, b, k, p) is a dominating node. Recurrences for $opt2(N, h, k, p)$ and $x(N, h, k, p)$ can be obtained by considering three cases for N. When N has no child (i.e., N is a leaf),

$$opt2(N, h, k, p) = \begin{cases} \infty & \text{if } k < 1 \text{ or } h < h(N, p) \\ 1 + s(N, p) & \text{otherwise} \end{cases} \quad (18)$$

$$x(N, h, k, p) = \begin{cases} \infty & \text{if } k < 1 \text{ or } h < h(N, p) \text{ or } N \text{ has an empty source trie} \\ opt2(N, h, k, p) & \text{otherwise} \end{cases} \quad (19)$$

[0110] When N has a single child a,

$$opt2(N, h, k, p) = \begin{cases} 1 + opt2(a, h-1-h(N, p)) + s(N, p) & \text{if } k = 1 \\ opt2(a, h, k-1, p+1) + s(N, p) & \text{otherwise} \end{cases} \quad (20)$$

$$x(N, h, k, p) = \begin{cases} \infty & \text{if } k < 1 \text{ or } h < h(N, p) \text{ or } N \text{ has an empty source trie} \\ opt2(N, h, k, p) & \text{otherwise} \end{cases} \quad (21)$$

[0111] When N has two children a and b,

$$opt2(N, h, k, p) = \begin{cases} 1 + opt2(a, h-1-h(N, p)) + opt2(b, h-1-h(N, p)) + s(N, p) & \text{if } k = 1 \\ \min \{ \begin{aligned} &opt2(a, h-1-h(N, p)) + opt2(b, h, k-1, p+1) + s(N, p), \\ &opt2(a, h, k-1, p+1) + opt2(b, h-1-h(N, p)) + s(N, p), \\ &\min_{0 < j < k-1} \{ opt2(a, h, j, p+1) + opt2(b, h, k-j-1, p+1) \} - 1 + s(N, p), \\ &\min x(N, h, k, p) \end{aligned} \} & \text{otherwise} \end{cases} \quad (22)$$

where

$$\min x(N, h, k, p) = \min_{0 < j < k-1} \{ x(a, h, j, p+1) + x(b, h, k-j-1, p+1) \} - 1 \quad (23)$$

$$x(N, h, k, p) = \begin{cases} \min x(N, h, k, p) & \text{if } N \text{ has an empty source trie} \\ opt2(N, h, k, p) & \text{otherwise} \end{cases} \quad (24)$$

[0112] Combining Equations 16 and 17, it can be obtained

$$opt2(N, h) = \min\{1 + ss(N) + \sum_{Q \in Ds(N)} opt2(Q, h-1 - h(Q)), \min_{0 < k \leq K} \{opt2(N, h, k, 0)\}\} \quad (25)$$

[0113] When there are n filters and the length of the longest prefix is W , the number of nodes in the dest trie of T and hence the number of exit tries is $O(nW)$. Using the algorithm of Section "Space-Optimal HSSTs", all $s(*, *)$ and $h(*, *)$ values may be computed in $O(n^2W^2HK^2)$ time. Following this computation, each $ss(N)$ value may be computed in $O(2^S) = O(K)$ time by traversing the first S levels of the subtree of T rooted at N . Thus all $ss(*)$ values may be determined in $O(nWK)$ additional time. As can be seen from Equation 25, $O(K)$ time is need to compute each $opt2(*, *)$ value (assuming that the ss and $opt2$ terms in the right-hand-side of the equation are known). It takes $O(K)$ time to compute each $opt2(*, *, *, *)$ and $x(*, *, *, *)$ value. As there are $O(nWH)$ $opt2(*, *)$ values and $O(nW^2HK)$ $opt2(*, *, *, *)$ and $x(*, *, *, *)$ values, the total time complexity is $O(n^2W^2HK^2 + nWK + nWHK + nW^2HK^2) = O(n^2W^2HK^2)$.

Implementation Considerations

HSSTs

[0114] If each supernode can be examined with a single memory access, then an HSST whose height is H (i.e., the number of levels is $H + 1$) may be searched for the next hop of the longest matching prefix by making at most $H + 2$ memory accesses. To get this performance, the supernode parameters K and S must be chosen such that each type of supernode can be retrieved with a single access. The size of a TBM node is $2^{S+1} + 2b - 1$ bits and that of an SST node is $4K + 2b - 1$ bits. An additional bit is needed to distinguish the two node types. So, any implementation of an HSST must allocate $2^{S+1} + 2b$ bits for a TBM node and $4K + 2b$ bits for an SST node. Such an implementation is referred to as the base implementation of an HSST. Let B be the number of bits that may be retrieved with a single memory access and suppose $b = 20$ bits is used for a pointer. When $B = 72$, the supernode parameters become $K = 8$ and $S = 4$. When $B = 64$, the supernode parameters become $K = 6$ and $S = 3$. Because of the need to align supernodes with word boundaries, each TBM node wastes 8 bits when $B = 64$.

[0115] An alternative implementation, called the prefix-bit implementation, has been proposed for supernodes. This alternative implementation employs the prefix-bit optimization technique. An additional bit (called prefixBit) is added to each supernode. This bit is a 1 for a supernode N if the search path through the parent supernode (if any) of N that leads to N goes through a binary trie node that contains a prefix. With the prefixBit added to each supernode, an HSST may be searched as follows:

Step 1: Move down the HSST keeping track of the parent, Z, of the most recently seen supernode whose prefixBit is 1. Do not examine the IBM of any node encountered in this step.

Step 2: Examine the IBM of the last supernode on the search path. If no matching prefix is found in this supernode, examine the IBM of supernode Z.

[0116] When prefix-bit optimization is employed, it is possible to have a larger K and S as the IBM (K or $2^S - 1$ bits) and NH (b bits) fields of a supernode are not accessed (except in Step 2). So, it is sufficient that the space needed by the remaining supernode fields be at most B bits. The IBM and NH fields may spill over into the next memory word. In other words, K and S are selected to be the largest integers for which $3K + b + 1 \leq B$ and $2^S + b + 2 \leq B$. When $B = 72$ and $b = 20$, $K = 17$ and $S = 5$ are used; and when $B = 64$ and $b = 20$, $K = 14$ and $S = 5$ are used. When prefix-bit optimization scheme is employed, the number of memory accesses for a search is $H + 4$ as two additional accesses (relative to the base implementation) are needed to fetch the up to two IBMs and NH fields that may be needed in Step 2.

[0117] The additional access to the IBM of Z may be avoided by using controlled leaf pushing. Recall that each supernode of an HSST represents a subtree of the binary trie T for the classifier. In controlled leaf pushing, the root N of the binary subtree represented by each supernode is examined. If N contains no next hop, the next hop of the longest prefix that matches $Q(N)$ is added to N. Note that when controlled leaf pushing is used, no prefixBit is needed and it is not necessary to keep track of the parent node Z during a lookup. This implementation of HSSTs with controlled leaf pushing is referred to as the enhanced prefix-bit implementation. The number of memory accesses required for a lookup in an enhanced prefix-bit implementation is $H + 3$.

Base Implementation Optimization

[0118] When the base implementation is used and $b = 20$, the value of K can be increased by 5 if the NH pointer (for a saving of b bits) can be eliminated. The elimination of the NH pointer may also lead to an increase in S . To eliminate the NH pointer, the next-hop array, NA , of a supernode N is stored next to its child array, CA . The start of the next-hop array for N can be computed from the child pointer of N and knowledge of the number of children supernodes that N has. The latter may be determined from the EBM of N . Since the size of a next-hop array may not be a multiple of B , this strategy may result in each next-hop array wasting up to $B - 1$ bits as each child array must be aligned at a word boundary. The total number of words of memory used by this enhanced base implementation can be reduced if some of the (CA, NA) pairs are paired and the second (CA, NA) tuple in each pair is flipped. For example, suppose that $B = 72$, each next-hop entry uses 18 bits, $NA1$ requires 162 bits, and $NA2$ requires 180 bits. Each entry in a child array is a supernode that uses B bits. Since each (CA, NA) must start at a word boundary, placing $(CA1, NA1)$ and $(CA2, NA2)$ into memory uses $n1 + n2 + 6$ B-bit words, where $n1$ and $n2$ are, respectively, the number of supernodes in $CA1$ and $CA2$. If $(CA2, NA2)$ is flipped to get $(NA2, CA2)$ then the next-hop array $NA2$ can use 36 of the 54 bits of a B-bit word not used by $NA1$ and reduce the total word count by 1. This sharing of a B-bit word by $NA1$ and $NA2$ leaves 18 unused bits in the shared B-bit word and the child array $CA2$ remains aligned to a word boundary. The child pointer for $(NA2, CA2)$ now points to the start of the array $NA2$ and to compute the start of the array $CA2$ from this child pointer, the number of next-hop entries in $NH2$ must be known. This number can be determined from the IBM. To employ this flipping strategy to potentially reduce the total memory required by the enhanced base implementation, each supernode must be augmented with a bit that identifies the orientation (CA, NA) or (NA, CA) used for its child and next-hop arrays.

[0119] To minimize the memory used by the enhanced base implementation, the following restricted bin packing problem (RBIN) must be solved: pack n integers $b_1, \dots, b_n$ in the range $[1, B]$, into the smallest number of size B buckets such that no bucket is assigned more than two of the integers. The RBIN problem may be solved in $O(n \log n)$ time by using the first-fit decreasing heuristic modified so as to pack at most two items in each bin. The optimality of this strategy is easily established by induction on n . An alternative strategy is to sort the b_i s into decreasing order and then to repeatedly pair the smallest unpaired b_i with the largest

unpaired b_i (under the constraint that the sum of the paired b_i s is no more than B). The pairing process terminates when no new pair can be created. The number of remaining singletons and pairs is the minimum number of bins needed for the packing.

End-Node Optimized HSSTs

[0120] A further reduction in the space requirements of an HSST may be achieved by employing end-node optimization. Four formats for a leaf supernode are permitted. FIG. 10 shows these four formats for the base implementation. Each supernode (leaf or non-leaf) uses a bit to distinguish between leaf and non-leaf supernodes. Each leaf supernode uses two additional bits to distinguish among the four leaf formats while each non-leaf supernode uses an additional bit to distinguish between SST and TBM supernodes. The leaf supernodes are obtained by identifying the largest subtrees of the binary trie T that fit into one of the four leaf-supernode formats. Notice that a leaf supernode has no child pointer. Consequently, in the SST format a larger K may be used than used for non-leaf supernodes and in the TBM format, a larger S may be possible. The third format (SuffixA) is used when the prefixes in a subtree are packed into a single supernode. For this packing, let N be the root of the subtree being packed. Then, $Q(N)$ (the prefix defined by the path from the root of T to N) is the same for all prefixes in the subtree rooted at N . Hence the leaf supernode need store only the suffixes obtained by deleting $Q(N)$ from each prefix in $ST(N)$. The leaf supernode stores the number of these suffixes, followed by pairs of the form (suffix length, suffix). In FIG. 10, $len(S_1)$ is the length of the first suffix and S_1 is the first suffix in the supernode. Leaf supernodes in the third format are searched by serially examining the suffixes stored in the node and comparing these with the destination address (after this is stripped of the prefix $Q(N)$; this stripping may be done as moving from $root(T)$ to N). The fourth format (SuffixB), which is similar to the third format, avoids the memory access required by the third format to extract the next hop. When controlled leaf pushing is applied to SuffixB supernodes, the worst-case number of memory accesses required for a lookup may decrease. Note that in the absence of controlled leaf pushing, if no matching prefix is found in a SuffixB leaf supernode, an additional access would be needed to extract the next hop associated with the longest matching prefix along the search path.

[0121] For all $ST(N)$ s may be represented by a leaf supernode of the first three types, set $opt(N, h) = 1$ for $h \geq 0$ and for all $ST(N)$ s that may be represented by a SuffixB supernode, set $opt(N, h)$

$= 1$ for $h \geq -1$. The dynamic programming recurrence of the Section "Space-Optimal HSSTs" is then used to determine $\text{opt}(\text{root}(T), H)$.

[0122] Although end-node optimization has only been described for the base implementation, this technique may be applied to the enhanced prefix-bit implementation as well to reduce total memory requirement.

2DHSSTs and 2DHSSTPCs

[0123] The enhanced base implementation of an HSST is used for both the dest and source tries of a 2DHSST and a 2DHSSTPC. End node optimization is done on each source trie of a 2DHSST and a 2DHSSTPC. For the dest trie, however, the following are done:

1. Cut off the leaves of the dest binary trie prior to applying the equations of the Sections "Space-Optimal 2DHSSTs" and "2DHSSTs With Prefix Inheritance (2DHSSTP)" to construct space-optimal 2DHSSTs and 2DHSSTPCs. Following the construction, identify the parent dest-trie supernode for each leaf that was cut off.

2. In the case of 2DHSSTPCs, each source trie that hangs off of a leaf of the dest binary trie, inherits the prefixes stored along the path, in the parent dest-trie supernode, to this leaf.

3. Each cut-off leaf is replaced by the HSST for its source trie (this source trie includes the inherited prefixes of (2) in case of a 2DHSSTPC). The root of this HSST is placed as the appropriate child of the parent dest-trie supernode. (This requires the use of an additional bit to distinguish between dest-trie supernodes and source HSST roots.)

[0124] By handling the leaves of the binary dest-trie as above, the need to search the source tries that are on the path, in the dest-trie parent, to a leaf child is eliminated.

[0125] Finally, for 2DHSSTPCs, the time and space required to construct space-optimal structures may be reduced by using an alternative definition of the p used in the Section "2DHSSTs With Prefix Inheritance (2DHSSTP)." In this new definition, prefix inheritance extends up to the p nearest ancestors of N in T that have a non-empty source trie. Since, on typical data sets, a dest-trie node has a small (say 3 or 4) number of ancestors that have non-empty source tries while the number of ancestors may be as large as 32 in IPv4 and 128 in IPv6, the new definition of p allows working with much smaller p s. This reduces the memory required by the arrays for $x(*, *, *, *)$ and $\text{opt2}(*, *, *, *)$ and also reduces the computation time. Note that the equations of the Section "2DHSSTs With Prefix Inheritance (2DHSSTP)" have to

be modified to account for this change in definition. Note also that while the space required for $\text{minx}(*, *, *, *)$ also is reduced, the recurrences of the Section "2DHSSTs With Prefix Inheritance (2DHSSTP)" may be solved without actually using such an array.

Experimental Results

[0126] C++ codes for the algorithms for space-optimal 1- and 2-dimensional supernode tries were compiled using the GCC 3.3.5 compiler with optimization level O3 and run on a 2.80 GHz Pentium 4 PC. The algorithms were benchmarked against recently published algorithms to construct space-efficient data structures for 1- and 2-dimensional packet classification. The benchmarked algorithms seek to construct lookup structures that (a) minimize the worst-case number of memory accesses needed for a lookup and (b) minimize the total memory needed to store the constructed data structure. As a result, the experiments measured only these two quantities. Further, all test algorithms were run so as to generate a lookup structure that minimizes the worst-case number of memory accesses needed for a lookup; the size (i.e., memory required) of the constructed lookup structure was minimized subject to this former constraint. For benchmarking purposes, it is assumed that the classifier data structure will reside on a QDRII SRAM, which supports both B=72 bits (dual burst) and B=144 bits (quad burst). For the experiments, b = 22 bits is used for a pointer (whether a child pointer or a pointer to a next-hop array) and 12 bits for each next hop. In the case of two-dimensional tables, the priority and action associated with a prefix need to be stored. 18 bits are allocated for this purpose.

One-Dimensional Routing Tables

[0127] Four variants of the space-optimal HSST were implemented: enhanced prefix-bit (EP), enhanced prefix-bit with end-node optimization (EPO), enhanced base (EB), and enhanced base with end-node optimization (EBO). In addition, the BFP algorithm of Song, et al. and the variant 3 algorithm (which is referred to as V3MT) of Sun, et al. are considered to construct multi-way trees. Extensive experiments reported in Sun, et al. establish the superiority of V3MT, in terms of space and lookup efficiency, over other known schemes for space and time efficient representation of IP lookup tables. Song, et al. establishes the superiority of BFP over TBM of Eatherton, et al. However, Song, et al. did not compare BFP to V3MT.

IPv4 Router Tables

[0128] For test data, both IPv4 and IPv6 router tables are used. First, the IPv4 experiments will be reported, which were conducted using the six IPv4 router tables Aads, MaeWest, RRC01, RRC04, AS4637 and AS1221. The number of prefixes in these router tables is 17486, 29608, 103555, 109600, 173501 and 215487, respectively.

[0129] FIG. 11 shows the number of memory accesses required for a lookup in the data structure constructed by each of the algorithms (assuming the root is held in a register) in the present invention. Unlike the access counts reported in Song, et al. and Sun, et al., the numbers reported by the present invention include the additional access (if any) needed to obtain the next hop for the longest matching prefix. FIG. 12 plots this data. As can be seen, EBO results in the smallest access counts for all of the test sets; EPO ties with EBO on all of the six test sets when $B = 72$ (other experiments with 9-bit next hop and 18-bit pointer fields indicate that EBO often requires one memory access less than EPO when $B=72$) and on 2 of the test cases when $B = 144$.

[0130] FIG. 13(a) normalizes the access count data by the counts for EBO and presents the min, max, and standard deviation of the normalized count for the 6 data sets. The number of memory accesses for a lookup in the structure constructed by BFP ranges from 1.33 to 2.00 times that required by the EBO structure; on average the BFP structure requires 1.53 times the number of accesses required by the EBO structure and the standard deviation is 0.25.

[0131] The number of memory accesses required by the structures constructed by each of the 6 test algorithms reduces when B goes from $B = 72$ to $B = 144$. The reduction for EPO is between 17% and 33% (the mean and standard deviation are 23% and 8%). The reduction for EBO is from 33% to 40% (the mean and standard deviation are 36% and 3%). Notice that when $B = 72$, BFP outperformed V3MT by 1 memory access on 5 of the 6 data sets and tied on the sixth. However, when $B = 144$, V3MT outperformed BFP by 1 memory access on 3 of the 6 data sets and tied on the remaining 3.

[0132] FIG.14 shows the total memory required by the lookup structure constructed by each of the 6 algorithms. FIG. 15 plots this data and FIG. 13(b) presents statistics normalized by the data for EBO. As can be seen, EPO and EBO result in the least total memory requirement. Although EPO is slightly superior to EBO on the memory measure on 9 of our 12 test cases, the total memory required by EBO for all 12 test cases is 2% less than that required by EPO. The search

structures constructed by the remaining algorithms required, on average, between 23% and 61% more memory than did the structures constructed by EBO. When $B = 72$, the average number of bits of storage needed per prefix is 48 for BFP, 42 for V3MT and 27 for EBO. The corresponding numbers for the case when $B = 144$ are 41, 35, and 27.

[0133] When B is increased from 72 to 144, the memory required by EPO and EBO decreased for 4 of the 6 data sets and increased for the remaining 2. The $B = 144$ memory normalized by the $B = 72$ memory is between .95 and 1.26, the average and standard deviation being 1.05 and .15, respectively. For EBO, the corresponding normalized numbers were .96, 1.13, 1.0, and .07.

[0134] On the IPv4 data sets, EBO and EPO are the clear winners. EBO is slightly superior to EPO on the memory access measure and the two are very competitive on the memory required measure. Since the former is more important, EBO is recommended over EPO. The EBO lookup structures require 25% to 50% fewer accesses than do the BFP structures; they also reduce memory requirement by 24% to 44%. The reduction in number of memory accesses and memory requirement relative to V3MT are 25% to 40% and 12% to 38%.

Comparison With Other Succinct Representations

[0135] One proposed a succinct router table structure is called Lulea. This is a 3-level multibit trie. A lookup in Lulea requires 12 memory accesses. So, as far as lookup time goes, Lulea is inferior to all 6 of the structures considered above. Since the code for Lulea is not available, only an approximate memory comparison is conducted. Degermark, et al. report memory requirements for 6 databases, the largest of which has 38,141 prefixes and uses 34 bits of memory per prefix. Since the memory required per prefix decreases as database size increases, it is compared with MaeWest, which has 29,608 prefixes (this comparison biases in favor of Lulea). On MaeWest, with $B = 72$, EPO, EBO, BFP and VM3T, respectively, require 32, 31, 55, and 49 bits per prefix. The corresponding numbers with $B = 144$ are 30, 35, 46, and 40. Note that this is a very approximate comparison for the following reasons (1) the databases are different and (2) the number of bits allocated to pointers and next hops is different in Lulea and the remaining structures. For example, the Lulea scheme requires the size of a pointer to be the same as that of a next hop and so allocates 14 bits to each. In the above experiments, 22 bits for a pointer and 12 for a next hop are used. Reducing the next hop size to 12 bits in Lulea doesn't

reduce the total memory required unless the pointer size is also reduced to 12 bits. Assuming these inequities balance out, the data suggest that EPO and EBO are superior to Lulea on both the lookup complexity and memory complexity!

[0136] Lunteren I has proposed a succinct representation of a multibit trie using perfect hash functions – balanced routing table search(BARTs). The first row of FIG. 16 gives the memory requirement of BARTs 12-4-4-4-8, one of his two most memory efficient schemes (the other scheme is BARTs 8-4-4-4-8, which requires slightly less memory but two more accesses for a search). The number of memory accesses needed for a lookup is 9 in BARTs 12-4-4-4-8. By comparison, the lookup complexity for EBO with $B = 72$ is 5 or 6 accesses/lookup, and the total memory required is between 38% and 43% the memory of BARTs 12-4-4-4-8. Note that the implementation assumptions used by Lunteren I and the present invention are slightly different. Lunteren I allocates 18 bits for each pointer and next hop whereas 22 bits for a pointer and 12 for a next hop are allocated in the present invention. The scheme of Lunteren I requires pointers and next hops to be of the same size. In reality, the number of different next hops is small and 12 bits are adequate. On the other hand, for large databases, 18 bits may not be adequate for a pointer. Despite these minor differences, the experiments show that EBO is superior to the scheme of Lunteren I on both lookup complexity and total memory required.

[0137] Lunteren II describes a related perfect-hash-function strategy (BARTS) for very wide memories, $B \geq 288$. The second row of FIG. 16 shows the memory requirement of his most memory efficient scheme, BARTS 12-6-6-8, for very wide memories. The reported data is for the case $B = 288$. The number of memory accesses needed for a lookup is 4. EBO with $B = 144$ achieves a lookup complexity of 3 or 4 accesses/lookup while requiring from 44% to 60% of the memory required by BARTS 12-6-6-8.

IPv6 Router Tables

[0138] For the IPv6 experiments, the 833-prefix AS1221-Telstra router table is used that as well as 6 synthetic IPv6 tables. Prefixes longer than 64 were removed from the AS1221-Telstra table as current IPv6 address allocation schemes use at most 64 bits. For the synthetic tables, the strategy proposed in Wang, et al. ("Non-random Generator for IPv6 Tables," 12th Annual IEEE Symposium on High Performance Interconnects, 2004) is used to generate IPv6 tables from IPv4 tables. In this strategy, a 16-bit string is prepended to each IPv4 prefix, which is comprised of

001 followed by 13 random bits. If this prepending doesn't at least double the prefix length, a sufficient number of random bits is prepended so that the length of the prefix is doubled. Following this prepending and possible appending, the last bit from one-fourth of the prefixes is dropped so as to maintain the 3:1 ratio of even length prefixes to odd length observed in real router tables. Each synthetic table is given the same name as the IPv4 table from which it was synthesized. The AS1221-Telstra IPv6 table is named AS1221* to distinguish it from the IPv6 table synthesized from the IPv4 AS1221 table.

[0139] FIGs. 17 and 20 give the number of memory accesses and memory required by the search structures for the 7 IPv6 data sets. FIGs. 18 and 21 plot these data and FIG. 19 gives statistics normalized by the data for EBO. EPO and EBO are the best with respect to number of memory accesses. When $B = 72$, EPO was superior to EBO by 1 memory access on 2 of the 7 data sets and tied on the remaining 5. However, when $B = 144$, EBO was superior to EPO by 1 memory accesses on 3 of the 7 data sets and tied on the remaining 4. As with the IPv4 data, the memory utilization of the EBO structures is almost as good as of the EPO structures (an average difference of 1%). Worst-case lookups in the constructed BFP structures require 1.14 to 2.00 times as many memory accesses as required in the EBO structures and the BFP structures require 1.82 to 3.17 times the memory required by the EBO structures.

[0140] As was the case for the IPv4 experiments, increasing B from 72 to 144, results in a reduction in the number of memory accesses required for a lookup. For EPO the maximum, minimum, and average reduction in the number of memory accesses were 33%, 17%, and 25%; the standard deviation was 8%. The corresponding percentages for EBO were 57%, 20%, 34%, and 13%. The memory required by EPO decreased for 5 of the 7 data sets while it increased on the remaining 2. The total memory required when $B = 144$ normalized by that required when $B = 72$ was between .86 and 1.1; the average and standard deviation were 1.01 and .1, respectively. For EBO, the memory required decreased on all of the 7 data sets. The maximum, minimum, and average memory reduction were 16%, 4%, and 12%; the standard deviation was 4%.

Further Optimizations

[0141] Song, et al. have proposed two techniques: child promotion and nearest-ancestor collapse, that may be used to reduce the number of nodes and number of prefixes in the one-bit

binary tree. These techniques reduce the size of the one-bit binary trie as well as that of its compact representation. In child promotion, the prefix stored in a binary node is promoted, if its sibling also contains a valid prefix, to the parent node. After the promotion, the node is deleted provided it is a leaf. In the nearest ancestor collapse technique, the prefix stored in a node is eliminated if its nearest ancestor contains a prefix with the same next hop; leaves are deleted if they become empty. Note that nearest-ancestor collapse is very similar to the port merge technique proposed by Sun, et al. A port merge is used to reduce the number of endpoints by merging two consecutive destination-address intervals that have the same next hop.

[0142] In this section, the effect of child promotion and nearest-ancestor collapse on the succinct representations generated by EBO, BFP, and V3MT will be studied. For V3MT, a port merge is done on the intervals constructed from the optimized binary trie. For this experimental study, only 3 of the IPv4 data sets - Aads, Maewest, and AS1221 are used as these are the only data sets for which next-hop data is available.

[0143] FIG. 22 gives the total memory requirement and memory accesses needed for a lookup. EBO remains the best succinct representation method on both the number of memory accesses measure and the total memory measure. On 2 of the 18 tests, (BFP on AS1221 with B=144 and V3MT on Aads with B=72), the number of memory accesses required for a lookup is reduced by 1. For the remaining 16 tests, there is no change in the number of accesses required for a lookup.

[0144] The application of the child promotion and nearest-ancestor collapse optimizations reduces the total memory required by the succinct representations of the binary trie. For EBO, the reduction varies from 24% to 35% with the mean reduction being 29%; the standard deviation is 5%. For BFP, these percentages were 23%, 37%, 28% and 6%. These percentages for V3MT were 34%, 49%, 40%, and 7%. The experiments indicate that most of the reduction in memory requirement is due to the nearest-ancestor collapse optimization. Child promotion contributed around 1% of the memory reduction.

[0145] The memory required by the BFP structures normalized by that required by the EBO structures was between 1.41 and 1.82, with the mean and standard deviation being 1.62 and 0.17. The corresponding ratios for V3MT were 1.02, 1.40, 1.19 and 0.16.

Multi-Dimensional IPv4 Tables

[0146] The performance of the proposed data structures using both 2-dimensional and 5-dimensional data sets is evaluated. Twelve 5-dimensional data sets that were created by the filter generator of Taylor, et al. are used. Each of these data sets actually has 10 different databases of rules. So, in all, there are 120 databases of 5-dimensional rules. The data sets, which are named ACL1 through ACL5 (Access Control List), FW1 through FW5 (Firewall), IPC1 and IPC2 (IP Chain) have, respectively, 20K, 19K, 19K, 19K, 12K, 19K, 19K, 18K, 17K, 17K, 19K, and 20K rules, on average, in each database. The 2-dimensional data sets, which were derived from these 5-dimensional data sets, have, respectively, 20K, 19K, 10K, 13K, 5K, 19K, 19K, 18K, 17K, 17K, 16K and 20K rules on average in each database. The 2-dimensional rules were obtained from the 5-dimensional rules by stripping off the source and destination port fields as well as the protocol field; the dest and source prefix field were retained. Following this stripping process, duplicates were deleted (i.e., two rules are considered duplicate if they have the same dest prefix and the same source prefix).

Two-Dimensional IPv4 Tables

[0147] First, the space-optimal minimum-access 2DHSST and 2DHSSTPC structures are compared. Figs. 23 and 24 show the results from the experiment. For 5 of the 12 data sets—ACL2-5, and IPC1—2DHSSTPCs reduce the number of accesses at the expense of increased memory requirement. For the remaining data sets, 2DHSSTPCs and 2DHSSTs require almost the same number of accesses and the same amount of memory.

[0148] Across all the data sets, 2DHSSTPCs required between 0% and 29% more memory than required by 2DHSSTs (the mean increase in memory required was 6% and the standard deviation was 9%). As noted earlier, although 2DHSSTPCs required more memory, they required a smaller number of memory accesses for a lookup. The reduction in number of memory accesses achieved by 2DHSSTPCs was between 0% and 41% (the mean reduction was 11% and the standard deviation was 13%).

[0149] When B is increased from 72 to 144, for both 2DHSSTs and 2DHSSTPCs, the number of memory accesses required is reduced, but the total memory required is generally increased. For 2DHSSTs, the total memory required when B = 144 normalized by that required when B = 72 is between 0.98 and 1.50 (the mean and the standard deviation are 1.21 and 0.19); the number of memory accesses reduces by between 28% and 41% (the mean reduction is 30% and the

standard deviation is 9%). For 2DHSSTPCs, the normalized memory requirement is between 1.04 and 1.49 (the mean and standard deviation are 1.23 and 0.16); the reduction in number of memory accesses ranges from 18% to 56% (the mean reduction and the standard deviation are 31% and 11%)

[0150] Since the primary objective is to reduce the number of memory accesses, 2DHSSTPCs with $B = 144$ for further benchmarking with 2DMTSas and 2DMTDs are used. The 2DMTSas and 2DMTDs used the compression techniques packed array and butler node. These two techniques are very similar; both attempt to replace a subtree with a small amount of actual data (prefixes and pointers) by a single node that contains these data. It is noted that 2DMTDs and 2DMTSas are the best of the structures developed, and using these two compression techniques, Lu, et al. have established the superiority of 2DMTDs and 2DMTSas over other competing packet classification structures such as Grid-of-Tries, EGT-PCs, and HyperCuts. For this further benchmarking, space-optimal 2DHSSTPCs with the minimum possible number, H , of memory accesses for a worst-case search are constructed. This minimum H was provided as input to the 2DMTSa (2DMTd) algorithm to construct a 2DMTSa (2DMTd) that could be searched with H memory accesses in the worst case. Because of this strategy, the worst-case number of memory accesses for 2DHSSTPCs and 2DMTSas (2DMTd) is the same.

[0151] Fig. 25 plots the memory memory required by 2DHSSTPCs, 2DMTDs, and 2DMTSas. It can be seen that on the memory criterion, 2DHSSTPCs outperform 2DMTSas by an order of magnitude, and outperform 2DMTDs by an order of magnitude on 4 of our 12 data sets. The memory required by 2DMTDs normalized by that required by 2DHSSTPCs is between 1.14 and 624, the mean and standard deviation being 56 and 179. The normalized numbers for 2DMTSas were 9, 49, 17, 11. It is also observed that when 2DMTDs are given up to 60% more memory than required by space-optimal DHSSTPCs with the minimum possible H , 2DMTDs that can be searched with 1 or 2 fewer accesses for the data sets FW1-5 and IPC2 can be constructed.

Five-Dimensional IPv4 Tables

[0152] For 5-dimensional tables, 2DHSSTPCs are extended using the bucket scheme proposed in Baboescu et al. ("Packet classification for core routers: is there an alternative to CAMs?," INFOCOM, 2003). Start with a 2-dimensional trie for the destination and source prefixes. All rules that have the same dest-source prefix pair (dp , sp) are placed in a bucket that is pointed at

from the appropriate source trie node of the 2-dimensional trie. Since dp and sp are defined by the path to this bucket, the dest and source prefix fields are not stored explicitly in a bucket. However, the source port range, dest port range, protocol type, priority and action are stored for each rule in the bucket. The 2DHSSTPC algorithms of this paper are used to obtain a supernode representation of the 2-dimensional trie and the NH lists of next-hop data are comprised of buckets. SuffixB nodes (an end-node optimization proposed in Lu, et al.) are modified so that they contain source prefix suffixes, dest and source ports, protocols, priorities and actions rather than just source prefix suffixes, priorities and actions. During prefix inheritance in 2DHSSTPCs, a source trie may inherit prefixes, from its ancestor tries, that already are in that source trie. When this happens, the rules associated with these inherited prefixes need also to be stored in this source trie. To avoid this redundancy, a pointer is stored in the bucket associated with a source-trie prefix, which points to the bucket associated with the same prefix in the nearest ancestor source trie. 2DHSSTPCs with buckets are called extended 2DHSSTPCs. Unlike 2DHSSTs, the source tries of an extended 2DHSSTPC are not modified so that the last source prefix seen on a search path has highest priority (or least cost).

[0153] Baboescu, et al. state that when 2-dimensional tries with buckets are used, as above, for 5-dimensional tables, most buckets have no more than 5 rules and no bucket has more than 20 rules. While this observation was true of the data sets used in Baboescu et al., some buckets had significantly more rules for our data sets. For example, in FW4, about 100 rules contain wildcards in both the dest and source prefix fields. These rules may be removed from the original data set and stored in a search structure that is optimized for the remaining 3 fields. It is noted that this strategy of storing a large cluster of rules with wildcards in the dest and source prefix fields in a separate structure was used earlier in the HyperCuts scheme. The data reported in the following figures and tables are only for structures constructed for the rules that remain after rules with wildcards in both dest and source prefix fields are removed.

[0154] Fig. 26 gives the total memory and number of memory accesses required by extended 2DHSSTPCs on the twelve 5-dimensional data sets. Fig. 27 compares 2DHSSTPCs (these, of course, store only the derived 2-dimensional rules) with extended 2DHSSTPCs that store 5-dimensional rules. The number of bits per rule required Memory (KBytes) by extended 2DHSSTPCs was between 59 and 196; the average was 128. Surprisingly, the addition of three fields increased the number of bits/rule by between 0.5 and 42 only; the average increase was

only 13. In fact, for 8 of the 12 data sets (ACL1-2, FW1-5, and IPC2), extended 2DHSSTPCs and 2DHSSTPCs (both using $B = 144$), required almost the same number of bits/rule. The very small increase in bits/rule is due to (a) port ranges in very many of our rules are wildcards that need only a bit each and (b) most of the data for the 3 additional fields in 5-dimensional rules can be packed into space in SuffixB nodes that is wasted in the case of 2-dimensional rules. The number of memory accesses required to search the extended 2DHSSTPCs ranged from 6 to 35; the average was 15. For 6 of our 12 data sets (ACL1, FW1-3, FW4, and IPC2), there was no increase in the number of memory accesses required for a lookup in an extended 2DHSSTPC for a particular 5-dimensional data set versus a lookup in the 2DHSSTPC for the corresponding 2-dimensional data set.

[0155] HyperCuts reported in Singh, et al., above, which is one of the previously known algorithmic schemes for multidimensional packet classification, uses a decision tree and rules are stored in buckets of bounded size; each bucket is associated with a tree node. Unlike the bucket scheme used by extended 2DHSSTPCs in which the dest and source prefixes are not stored explicitly, the bucket scheme of HyperCuts requires the storage of these fields as well as those stored in extended 2DHSSTC buckets. So, the storage of an individual rule in HyperCuts requires more space than is required in extended 2DHSSTPCs. Additionally, in HyperCuts, a rule may be stored in several buckets whereas in extended 2DHSSTPCs, each rule is stored in exactly 1 bucket. The most efficient Hypercut scheme is HyperCuts-4. This scheme is used for comparison with extended 2DHSSTPCs.

[0156] Fig. 28 shows the total memory and number of memory accesses required by HyperCuts, on the twelve 5-dimensional data sets. The number of bits per rule required by the HyperCuts structure was between 242 and 163,519; the average was 56,801. It is important to note that there is wide variation in the bits/rule required by Hypercuts; the bits/rule required by extended 2DHSSTPCs is far better predictable. In particular, Singh, et al. report that the performance of HyperCuts is not good for firewall-like databases as these tend to have a high frequency of wildcards in the source and/or dest fields. In fact, Singh, et al., above, report that a 10% presence of wildcards in either the source or dest prefix fields resulted in a steep increase in memory requirement. This observation is confirmed by the experiments in the present invention. HyperCuts exhibited its best bits/rule performance on ACL1 and ACL5 (242 and 400, respectively), in which the frequency of wildcards in either the source or dest fields is less than

1%. It exhibited its worst performance on the 5 firewall data sets FW1-5 and on IPC2 (bits/rule ranged from 25,757 to 163,519). The wildcard frequency was between 60% and 90% in these data sets. The remaining data sets (ACL2-4 and IPC1) had a wildcard frequency between 10% and 15% and the bits/rule required by the Hypercuts structure varied from 4,415 to 16,363. The number of accesses required to search the Hypercuts structure for the data sets ranged from 16 to 51, with the average being 30.

[0157] Fig. 29 compares extended 2DHSSTPCs and HyperCuts. The structure constructed by extended 2DHSSTPCs required between 0.1% and 79% the memory required by that constructed by HyperCuts; the average and standard deviation being 8% and 23%, respectively. The number of accesses for a lookup in the extended 2DHSSTPCs structure was between 31% and 81% that required by the HyperCuts structure; the average and standard deviation were 46% and 16%, respectively. For both schemes, the reported memory and accesses are only for the rules that remain after rules with wildcards in both dest and source prefix fields are removed.

[0158] Since, in extended 2DHSSTPCs, no rule is stored twice while the same rule may be stored in several Hypercuts buckets (depending on the complexity of the rule set), the memory requirement of 2DHSSTPCs is better predicted and far less on average and worst-case data.

[0159] Referring now to FIG. 30, a method 300 for classifying packets of data transmitted over a data communications network is represented by the flowchart of exemplary steps. Illustratively, after initial step 302, a memory-storable packet classifier rule defined by a plurality of filters is provided, at step 304. Each filter uniquely corresponds to one of a plurality of actions to be taken in transmitting the data packets over the data communications network. At step 306, the a data packet is classified according to the packet classifier rule by generating at least one supernode trie subject to a predetermined constraint for reducing accesses to a memory in which the packet classifier rule is stored. The method illustratively concludes at step 308.

[0160] According to one embodiment, the generating step 306 comprises generating a 1-dimensional node, space-optimal supernode, wherein the predefined constraint minimizes the memory accesses. According to another embodiment, the generating step 306 comprises generating a mult-dimensional node, space-optimal supernode, wherein the predefined constraint minimizes the memory accesses. Moreover, the supernode and predefined constraint can be configured to minimize an amount of memory for storing the at least one classifier rule.

[0161] The invention, as already noted, can be realized in hardware, software, or a combination of hardware and software. The invention can be realized in a centralized fashion in one computer system, or in a distributed fashion where different elements are spread across several interconnected computer systems. Any kind of computer system or other apparatus adapted for carrying out the methods described herein is suited. A typical combination of hardware and software can be a general purpose computer system with a computer program that, when being loaded and executed, controls the computer system such that it carries out the methods described herein.

[0162] The invention can be embedded in a computer program product, or computer-readable storage medium, which comprises all the features enabling the implementation of the methods described herein, and which when loaded in a computer system is able to carry out these methods. Computer program in the present context means any expression, in any language, code or notation, of a set of instructions intended to cause a system having an information processing capability to perform a particular function either directly or after either or both of the following: a) conversion to another language, code or notation; b) reproduction in a different material form.

[0163] The invention can be embodied in other forms without departing from the spirit or essential attributes thereof. Accordingly, reference should be made to the following claims, rather than to the foregoing specification, as indicating the scope of the invention.

CLAIMS

We claim:

1. A computer-implemented method of classifying packets of data transmitted over a data communications network, the method comprising:
providing a memory-storable packet classifier rule defined by a plurality of filters, each filter uniquely corresponding to one of a plurality of actions to be taken in transmitting the data packets over the data communications network; and
classifying a data packet according to the packet classifier rule by generating at least one supernode trie subject to a predetermined constraint for reducing accesses to a memory in which the packet classifier rule is stored.
2. The method of Claim 1, wherein the generating step comprises generating a 1-dimensional node, space-optimal supernode, wherein the predefined constraint minimizes the memory accesses.
3. The method of Claim 1, wherein the generating step comprises generating a multi-dimensional node, space-optimal supernode, wherein the predefined constraint minimizes the memory accesses.
4. The method of Claim 1, wherein the supernode and predefined constraint are configured to minimize an amount of memory for storing the at least one classifier rule.
5. A system comprising a combination of hardware and software modules for classifying packets of data transmitted over a data communications network, the system comprising:
a memory for storing at least one packet classifier rule defined by a plurality of filters, each filter uniquely corresponding to a one of a plurality of actions to be taken in transmitting the data packets over the data communications network; and
a processor configured to access the packet classifier rule in the memory and to classify a data packet based upon the packet classifier rule, the data packet being classified by the

processor generating at least one supernode trie subject to a predetermined constraint for reducing accesses to the memory.

6. The system of Claim 5, wherein the classifier rule comprises a 1-dimensional node, space-optimal supernode and wherein the predefined constraint minimizes memory accesses performed by the processor in classifying the data packet.

7. The system of Claim 5, wherein the classifier rule comprises a multi-dimensional node, space-optimal supernode and wherein the predefined constraint minimizes memory accesses performed by the processor in classifying the data packet.

8. The system of Claim 5, wherein the processor is configured to generate a supernode that corresponds to a predefined constraint that minimizes storage of the at least one classifier rule within the memory.

9. A computer-readable storage medium for storing a computer program comprising computer-implemented instructions for:

accessing a memory-storable packet classifier rule defined by a plurality of filters, each filter uniquely corresponding to one of a plurality of actions to be taken in transmitting the data packets over the data communications network; and

classifying a data packet according to the packet classifier rule by generating at least one supernode trie subject to a predetermined constraint for reducing accesses to a memory in which the packet classifier rule is stored.

10. The computer-readable storage medium of Claim 9, wherein the generating step comprises generating a 1-dimensional node, space-optimal supernode, wherein the predefined constraint minimizes the memory accesses.

11. The computer-readable storage medium of Claim 9, wherein the generating step comprises generating a multi-dimensional node, space-optimal supernode, wherein the predefined constraint minimizes the memory accesses.

12. The computer-readable storage medium of Claim 9, wherein the supernode and predefined constraint are configured to minimize an amount of memory for storing the at least one classifier rule.

1/18

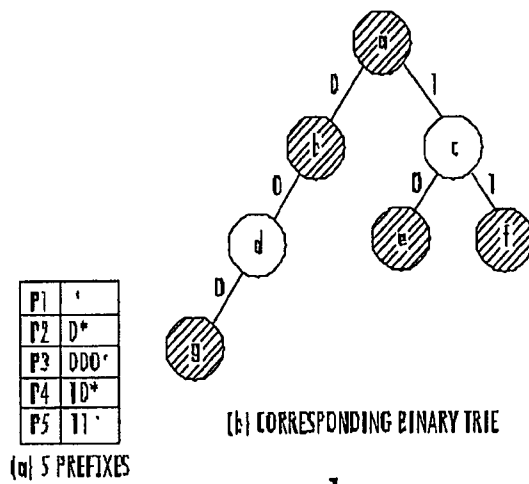


FIG. 1

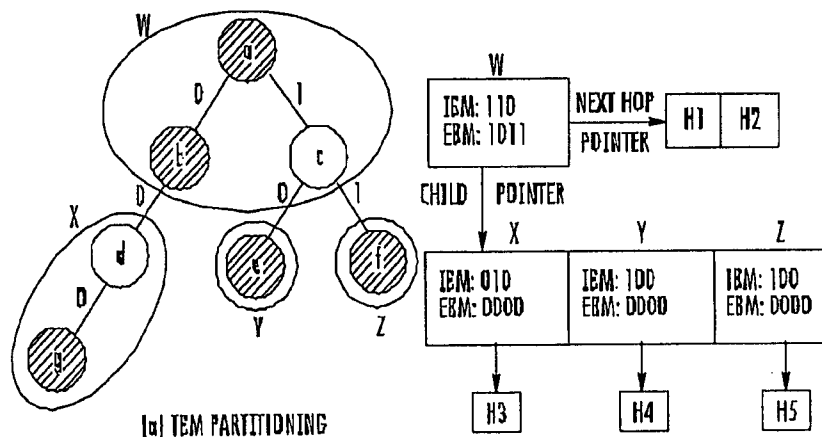


FIG. 2

2/18

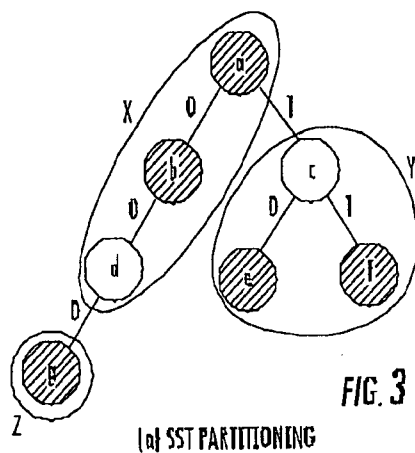


FIG. 3

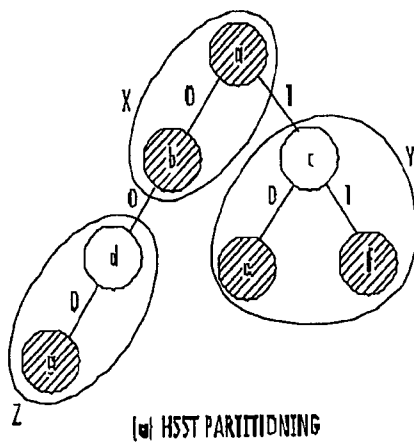
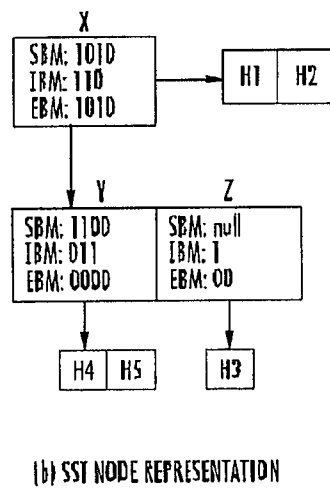
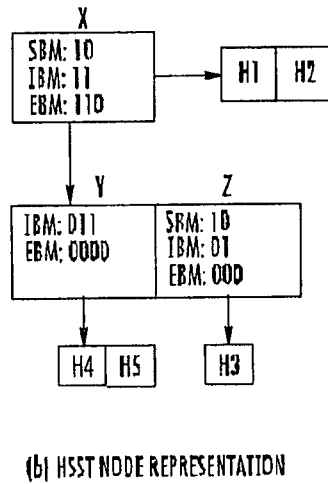


FIG. 4



3/18

FILTER	DEST PREFIX	SOURCE PREFIX	COST
F1	-	0*	1
F2	0*	1*	2
F3	000*	0*	3
F4	10*	0*	4
F5	11*	1*	5

AN EXAMPLE OF FIVE DEST-SOURCE FILTERS

FIG. 5

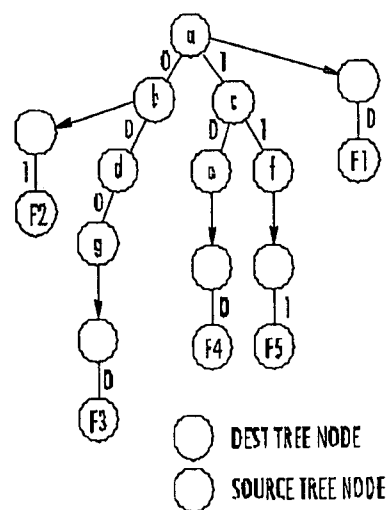


FIG. 6

4/18

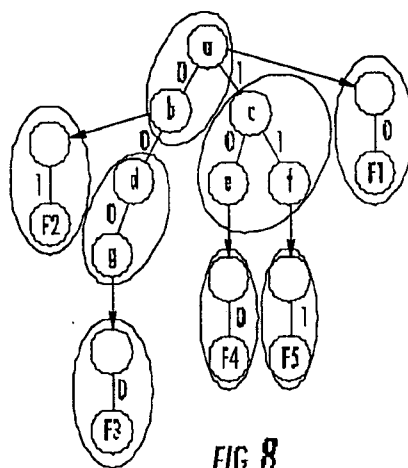


FIG. 8

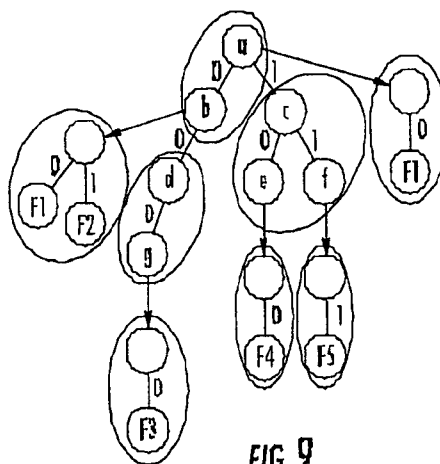


FIG. 9

5/18

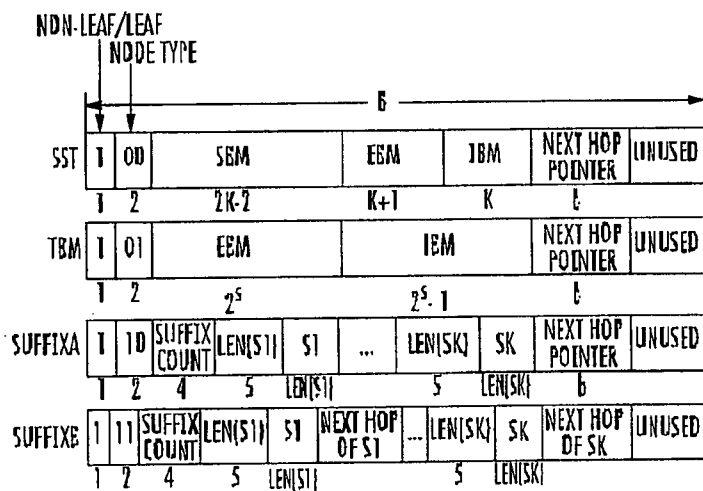


FIG. 10

DATABASE	B=72 BITS						B=144 BITS					
	EP	EPO	EE	EBO	BFP	V3MT	EP	EPD	EB	EBO	BFP	V3MT
AADS	6	5	6	5	7	8	5	4	4	3	6	5
MAEWEST	6	5	6	5	7	8	5	4	4	3	6	5
RRC01	6	6	6	6	8	8	5	4	5	4	6	6
RRC04	7	6	6	6	8	9	5	4	5	4	6	6
AS4637	7	6	6	6	8	9	5	5	5	4	6	6
AS1221	7	6	7	6	8	9	5	5	5	4	7	6

FIG. 11

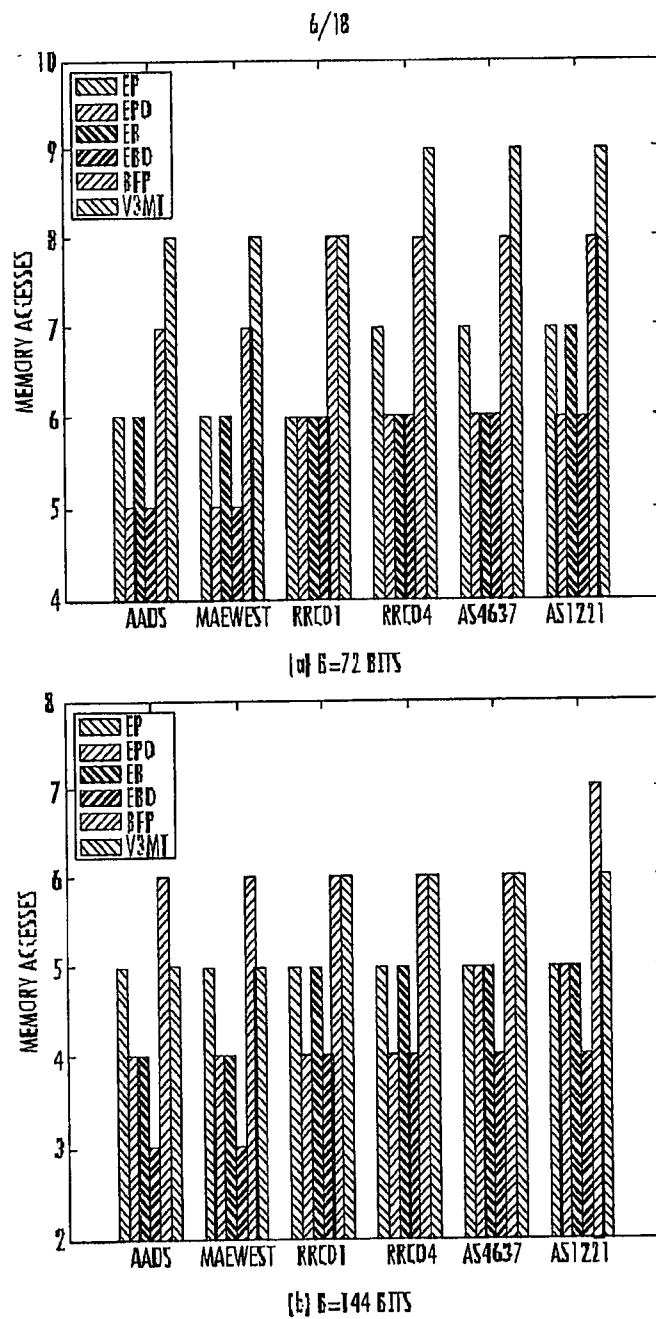


FIG. 12

7/18

ALGORITHM	MIN	MAX	MEAN	STANDARD DEVIATION
EP	1.00	1.67	1.27	0.20
EPO	1.00	1.33	1.10	0.15
EE	1.00	1.33	1.19	0.12
EBO	1	1	1	0
BFP	1.33	2.00	1.53	0.25
V3MT	1.33	1.67	1.53	0.09

(a) ACCESS

FIG. 13

(b) MEMORY

DATABASE	B=72 BITS						B=144 BITS					
	EP	EPO	EB	EBO	BFP	V3MT	EP	EPO	EB	EBO	BFP	V3MT
AADS	102	67	92	68	122	110	86	64	81	71	103	89
MAEWEST	167	116	150	113	199	180	141	111	135	128	167	146
RKCD1	479	326	428	335	578	515	419	405	382	214	488	427
RKCD4	484	344	452	354	610	544	444	432	404	333	514	452
AS4637	721	518	682	530	918	798	677	493	611	507	785	663
AS1221	870	620	810	644	1108	981	823	596	750	635	957	819

FIG. 14

8/18

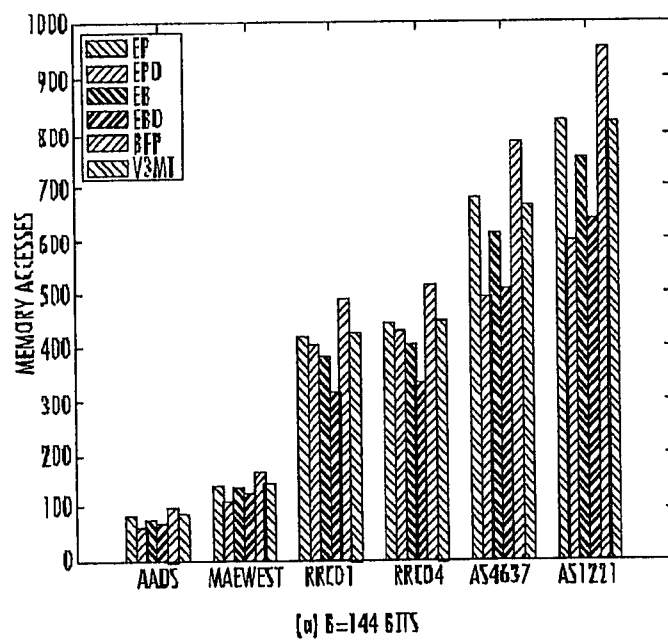
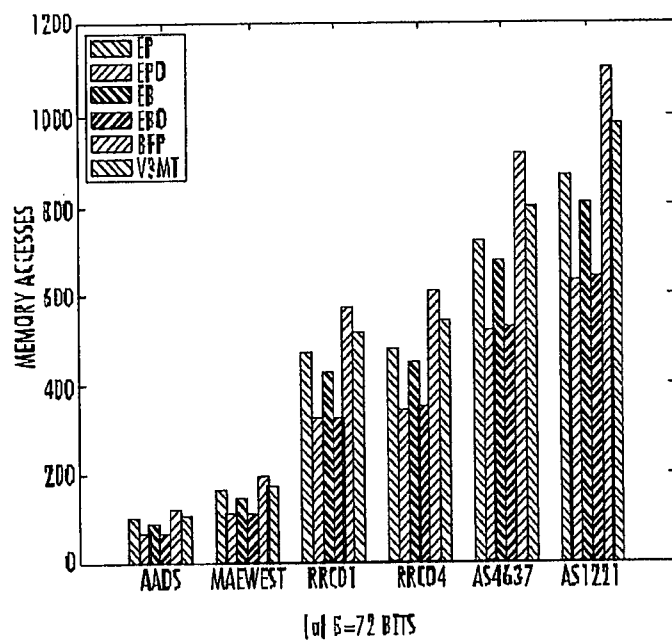


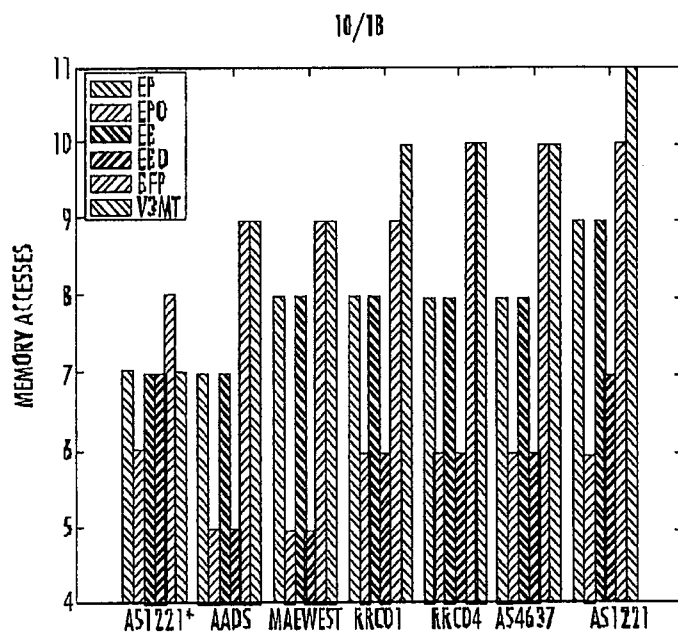
FIG. 15

DATABASE		AADS	MAEWEST	RRCD1	RRCD4	AS4637	AS1221
BARTS	12-4-4-8	[E=32]	163	793	844	1270	1685
	12-6-6-8	[E=288]	137	692	733	1149	1380

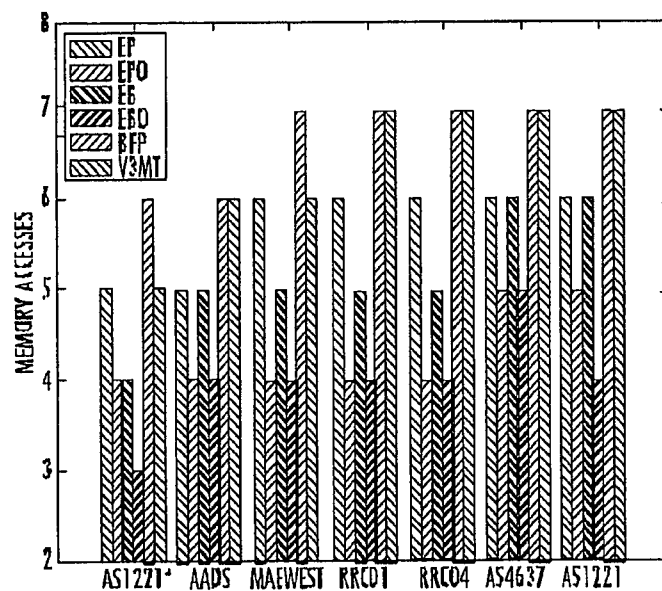
FIG. 16

DATABASE	B=72 BITS							B=144 BITS						
	EP	EPO	EE	EED	BFP	V3MT		EP	EPD	EB	EBO	BFP	V3MT	
AS1221	7	6	7	7	8	7		5	4	4	3	6	5	
AADS	7	5	7	5	9	9		5	4	5	4	6	6	
MAEWEST	8	5	8	5	9	9		6	4	5	4	7	6	
RRCD1	8	6	8	6	9	10		6	4	5	4	7	7	
RRCD4	8	6	8	6	10	10		6	4	5	4	7	7	
AS4637	8	6	8	6	10	10		6	5	6	4	7	7	
AS1221	9	6	9	7	10	11		6	5	6	4	7	7	

FIG. 17



(a) B=72 BITS



(a) B=144 BITS

FIG. 18

11/18

ALGORITHM	MIN	MAX	MEAN	STANDARD DEVIATION
EP	1.42	2.99	2.47	0.45
EPD	0.92	1.21	0.99	0.10
EE	1.35	2.16	1.97	0.26
EBO	1	1	1	0
BFP	1.82	3.17	2.71	0.42
V3MT	2.26	3.12	2.79	0.30

ALGORITHM	MIN	MAX	MEAN	STANDARD DEVIATION
EP	1.00	1.67	1.41	0.17
EPD	0.86	1.33	1.04	0.14
EE	1.00	1.60	1.33	0.14
EBO	1	1	1	0
BFP	1.14	2.00	1.66	0.21
V3MT	1.00	1.80	1.63	0.21

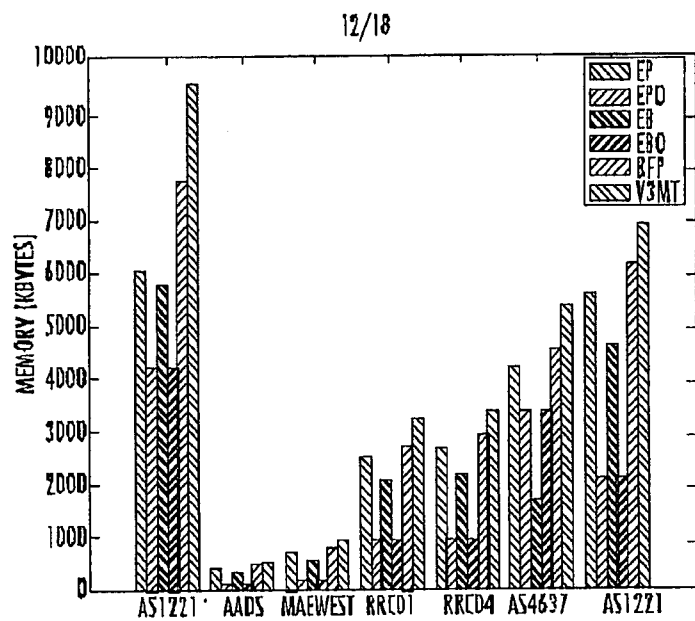
(b) MEMORY

(a) ACCESS

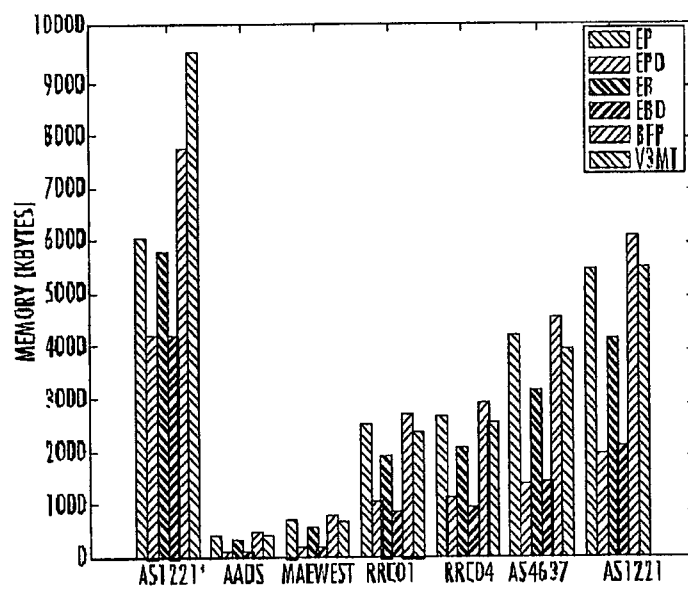
FIG. 19

DATABASE	B=72 BITS						B=144 BITS					
	EP	EPD	EE	EBO	BFP	V3MT	EP	EPD	EE	EBO	BFP	V3MT
AS1221	7255	4372	6574	4755	8634	10996	6043	4213	5760	4254	7729	9601
AADS	473	170	391	184	501	558	464	157	334	155	492	438
MAEWEST	769	296	641	311	836	946	750	284	563	270	832	724
RRC01	2581	974	2102	1046	2785	3267	2556	1107	1969	917	2782	2405
RRC04	2744	1030	2234	1108	2969	3459	2712	1163	2098	971	2949	2562
AS4637	4249	1617	3449	1752	4609	5445	4216	1481	3201	1503	4584	3957
AS1221	5656	2182	4665	2252	6240	6982	5489	2005	4220	2168	6090	5516

FIG. 20



(a) 8=72 BITS



(b) 8=144 BITS

FIG. 21

13/18

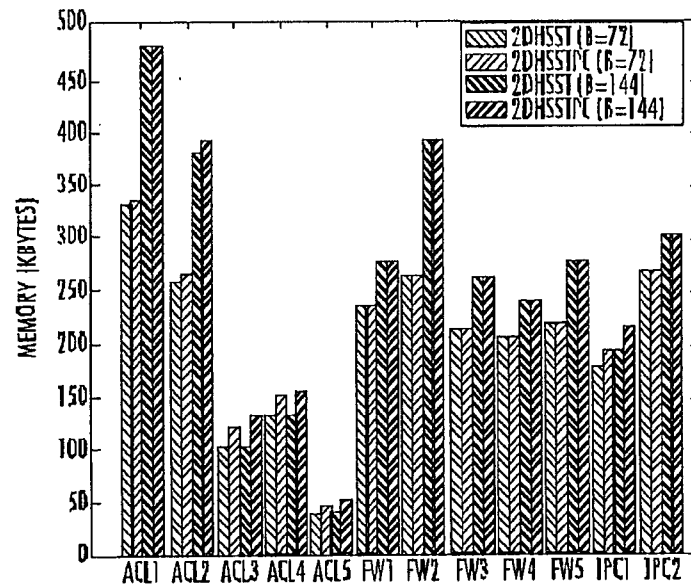
DATABASE	E=72 BITS						E=144 BITS					
	EBD		BFP		V3MT		EBD		BFP		V3MT	
	MEM	MA _s	MEM	MA _s	MEM	MA _s	MEM	MA _s	MEM	MA _s	MEM	MA _s
AADS	51	5	94	7	72	7	50	3	78	6	57	5
MAEWEST	85	5	151	7	116	8	89	3	126	6	93	5
AST221	425	8	716	8	507	9	411	4	605	8	418	8

FIG. 22

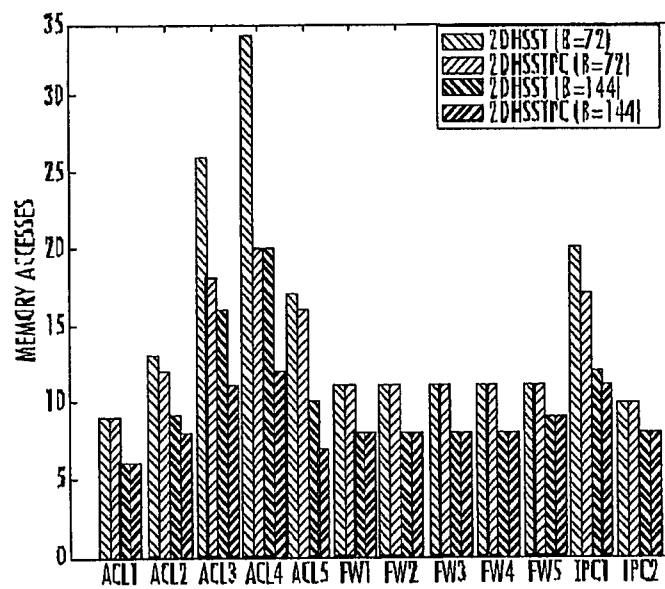
DATA SET	E=72 BITS				E=144 BITS			
	2DHSST		2DHSSTPC		2DHSST		2DHSSTPC	
	MEM	MA _s	MEM	MA _s	MEM	MA _s	MEM	MA _s
ACL1	329	9	333	9	479	6	479	6
ACL2	256	13	264	12	379	9	388	8
ACL3	102	26	119	18	102	16	131	11
ACL4	130	34	148	20	131	20	154	12
ACL5	40	17	45	16	39	10	50	7
FW1	236	11	236	11	274	8	274	8
FW2	261	11	261	11	392	8	390	8
FW3	210	11	210	11	260	8	260	8
FW4	203	11	203	11	237	8	237	8
FW5	216	11	216	11	274	9	273	9
IPC1	174	20	190	17	189	12	211	11
IPC2	265	10	265	10	300	8	300	8

FIG. 23

14/18



(a) MEMORY



(b) ACCESSSES

FIG. 24

15/18

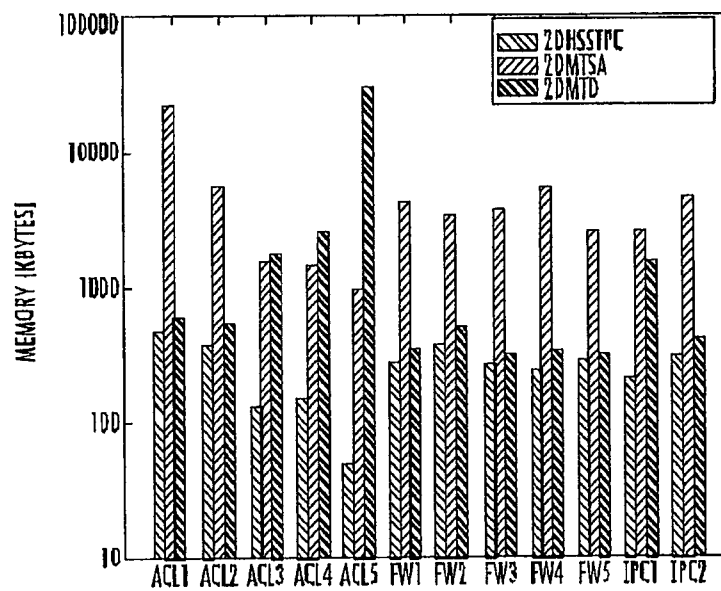


FIG. 25

DATA SET	MEM	MA _s	BITS/RULE
ACL1	480	6	196
ACL2	392	12	169
ACL3	181	35	78
ACL4	252	32	108
ACL5	87	20	59
FW1	282	8	121
FW2	391	8	168
FW3	264	8	120
FW4	302	9	145
FW5	280	9	135
IPC1	244	23	105
IPC2	326	8	133

FIG. 26

16/18

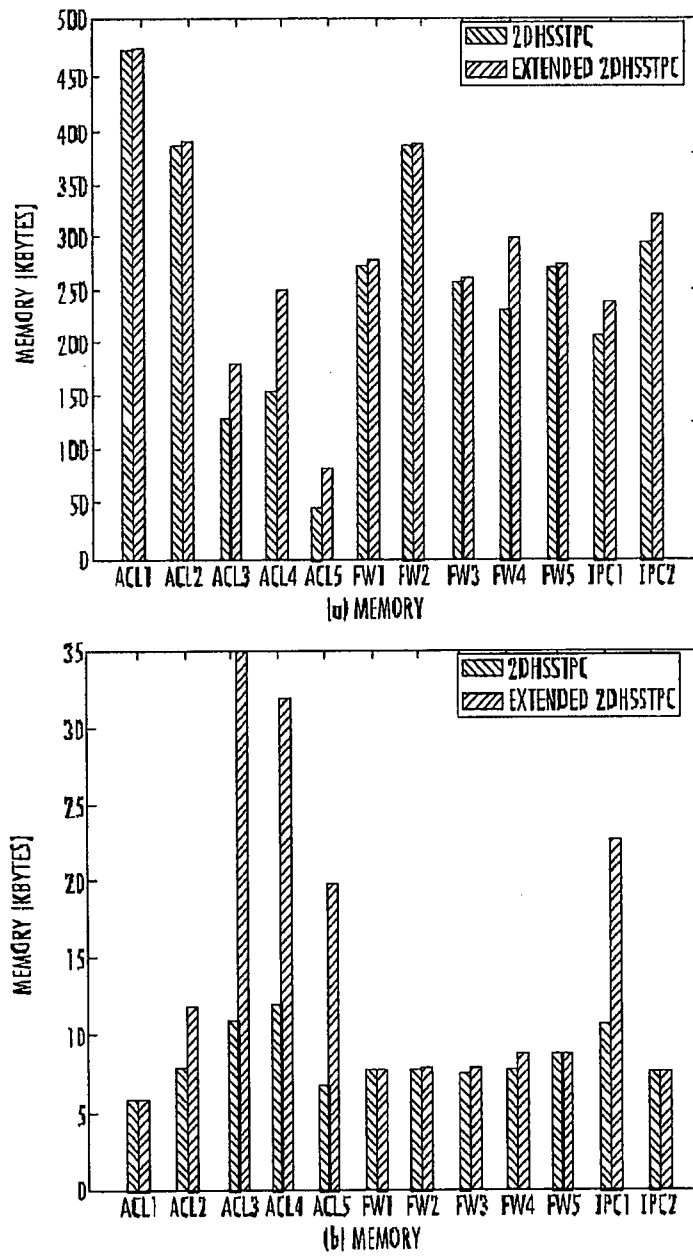


FIG. 27

17/18

DATA SET	MEM	MA _s	BITS/RULE
ACL1	605	16	242
ACL2	10487	24	4415
ACL3	19591	43	8248
ACL4	17661	44	7436
ACL5	600	44	400
FW1	308121	26	129735
FW2	189751	24	79895
FW3	341813	23	151916
FW4	199096	24	93692
FW5	347478	20	163519
IPC1	38863	51	16363
IPC2	64394	24	25757

FIG. 28

18/18

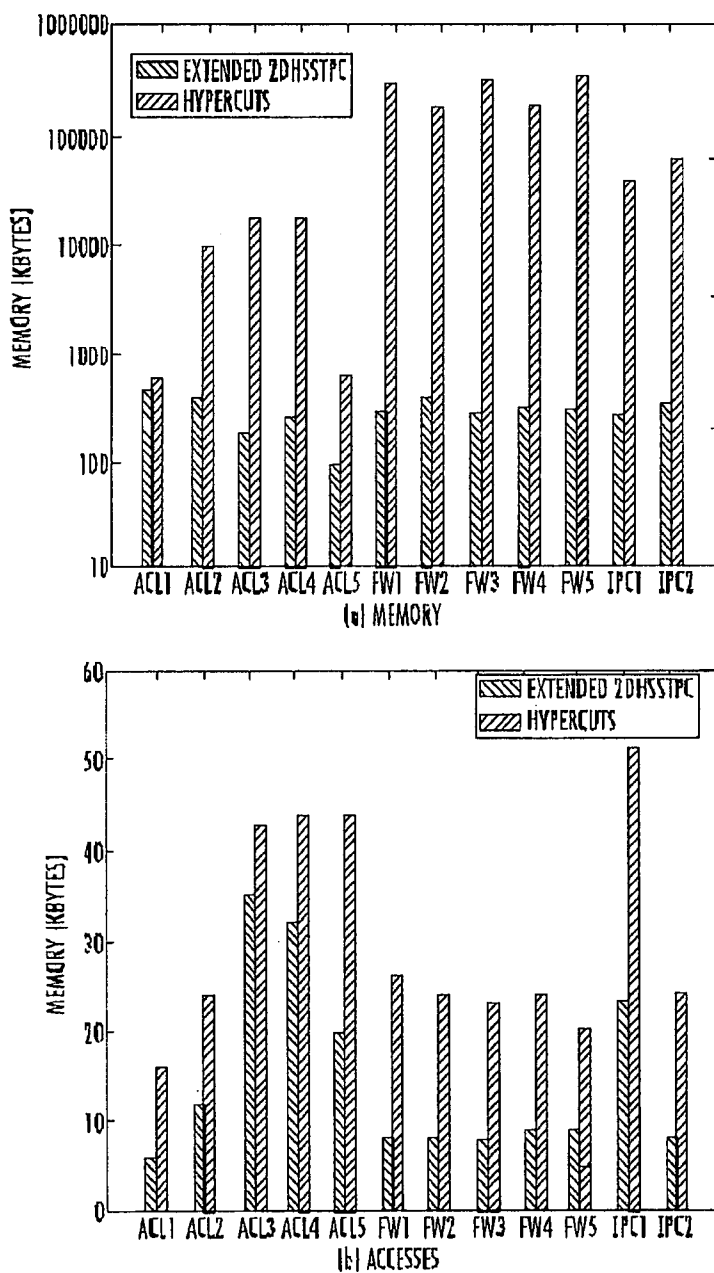


FIG. 29