ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.9, September 2006, pp.1949−1957
DOI: 10.1360/jos171949

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种适合于网络处理器的并行多维分类算法 AM-Trie *

郑 波+, 林 闯, 曲 扬

(清华大学 计算机科学与技术系,北京 100084)

AM-Trie: A Parallel Multidimensional Packet Classification Algorithm Fitting for Network Processor

ZHENG Bo+, LIN Chuang, QU Yang

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

+ Corresponding author: Phn: +86-10-62772487, Fax: +86-10-62771138, E-mail: bzheng@csnet1.cs.tsinghua.edu.cn

Zheng B, Lin C, Qu Y. AM-Trie: A parallel multidimensional packet classification algorithm fitting for network processor. *Journal of Software*, 2006,17(9):1949−1957. http://www.jos.org.cn/1000-9825/17/1949.htm

Abstract: Nowadays, many high speed Internet applications require high speed multidimensional packet classification algorithms. Based on the uniqueness of Network Processor, this paper presents a multidimensional classification algorithm—AM-Trie (asymmetrical multi-bit trie). AM-Trie is a high speed, parallel and scalable algorithm and very fit for the "multi-thread and multi-core" feature of the Network Processor. A heuristic field division algorithm is also presented, and it is proved theoretically that it can find out the minimum storage cost solution when the height of the AM-Tire is given. Finally, a prototype is implemented based on Intel IXP 2400 Network Processor. The performance testing result shows that AM-Trie is a high-speed and scalable algorithm; the throughput of the whole system is influenced little by the size of rules and it can reach 2.5 Gbps wire speed.
Key words: packet classification; network processor; parallel algorithm; multidimensional; AM-Trie

摘 要: 针对当前高速网络应用对分组分类算法的要求以及网络处理器体系结构的特点,提出了一种高速多维分组分类算法——AM-Trie 算法(asymmetrical multi-bit trie,非对称多权 Trie 树).该算法具有搜索速度快,并行性、可扩展性良好的特点,特别适合于在网络处理器上实现.同时,给出了一种空间最优的启发式分类字段分段算法,并从理论上证明其在确定 AM-Trie 树层数的情况下使得存储空间最小.最后,基于 Intel IXP2400 网络处理器设计并实现了该算法.性能实测表明,该算法性能良好并具有很好的可扩展性,算法速度受规则库大小的影响很小,在各种情况下均达到了 2.5Gbps 的线速.
关键词: 分组分类;网络处理器;并行算法;多维分类;AM-Trie
中图法分类号: TP393 文献标识码: A

* Supported by the Major Research Plan of the National Natural Science Foundation of China under Grant No.90412012 (国家自然科学基金重大研究计划重点项目); the National Grand Fundamental Research 973 Program of China under Grant No.2003CB314804 (国家重点基础研究发展规划(973)); the Juniper Research Grant (Juniper 公司研究基金); the Intel IXA University Research Plan (Intel IXA 大学研究计划)

Received 2005-10-27; Accepted 2006-01-09

目前,计算机网络正处在蓬勃发展的时期,网络速度不断提高,用户数量急剧增长.同时,Internet 服务也开始由原先的尽力服务(best-effort service)向 QoS(quality of service)发展,网络设备(路由器、交换机等)的功能必须由原先单纯的转发分组提升到具有内容知晓(content-awareness)的能力,而分组分类则是其中重要的一环.许多网络关键技术,如虚拟专用网(VPNs)、网络地址转换(NAT)、防火墙、网络入侵检测(IDS)、QoS、拥塞控制、组播、MPLS 通道的流合并以及未来的 IPv6 流标识等都涉及到分组分类.因此,分组分类速度的快慢、功能的强弱将直接影响到许多网络技术的性能,并且对下一代网络及其服务质量有关键性的影响.由此可见:分组分类是现今网络研究的重要议题之一,Sigcomm,Infocom 等重要国际会议近几年均有相关文章发表[1−5].

网络处理器是在网络速度继续提高、网络应用和服务日益多样化的情况下,为同时满足高速处理、高灵活性和短研制周期而出现的一种新兴技术.它被认为是推动下一代网络发展的核心技术.基于网络处理器平台的算法实现可以轻松地扩展服务类型或移植到其他系统上,这对于开发商和用户而言,无疑都是一个很好的选择.网络处理器具有很好的发展前景,其相关技术的研究日益成为一个热门课题[6−8].

因此,在网络处理器这种新技术平台上设计和实现高速分组分类算法,无论是从技术角度还是从市场角度都具有十分重要的意义.当前的高速分组分类通常用硬件(例如 TCAM 和 FPGA)来实现,但是,这类的硬件成本非常高(特别是在比特长度和容量都很大的情况下)、功耗大,因此,业界都希望能用软件的方法达到高速分组分类的效果.以往的分组分类算法通常是用二叉树或者是哈希表的数据结构,根据规则库的一些统计特点构造查找树或哈希表.这类算法的复杂度为 $O(DW)$,其中:D 为分类的维数;W 是分类字段的比特数.对于目前的高速网络,应用这类算法已经显得力有未逮,更不用说将来的高速 IPv6 应用了.此外,这些算法通常不能并行执行,不能发挥网络处理器多内核、多硬件线程的特点.

本文基于网络处理器高度并行的特点设计了一种可并行执行的高速多维分类算法——AM-Trie (asymmetrical multi-bit trie,即非对称多权 Trie 树)算法.该算法的主要特点如下:

- 创新地利用冗余表示任意长度的前缀,通过 AM-Trie 树来降低搜索的复杂度;
- 通过压缩 AM-Trie 树消除回溯,进一步提高搜索速度并减少存储空间,AM-Trie 算法时间复杂度为 $O(d+h)$,d 为分类的维数,h 为 AM-Trie 树的高度,空间复杂度为 $O(N^2)$,N 为分类规则数;
- 从理论上给出一种启发式算法来设计 AM-Trie 树每一层的宽度,在确定 AM-Trie 树层数(时间复杂度)的情况下,达到存储空间(空间复杂度)最小,并给出优化过程的复杂度;
- 算法可并行执行,很好地利用了网络处理器多内核、多硬件线程的特点;
- 算法的可扩展性良好,规则数的增加对算法性能几乎没有影响.

在此基础上,我们基于 Intel 网络处理器 IXP2400 (最高速率为 OC48)实现了该算法,性能分析和测量结果表明,我们的分组分类算法在 TCP/IP 六元组分类中性能达到了 2.5Gbps 的线速(每个分组大小为 64 字节,亦即 5Mpps).

本文第 1 节介绍网络处理器的特点.第 2 节详细介绍 AM-Trie 算法.第 3 节提出并证明 AM-Trie 的空间优化算法.第 4 节对算法进行性能测量和分析.最后,第 5 节对全文进行总结.

1 网络处理器简介

网络处理器是继 ASIC 之后一个新的网络技术发展方向和研究热点,主要具有以下技术特点:

(1) 并行处理机制.网络处理器内部通常是多内核(multi-core)的结构[9],这些内核一般采用以下两种组织机制.流水线:每个内核具有特定处理功能,这些内核以流水线的方式组织在一起完成分组的处理,这种结构的网络处理器主要有 Cisco 的 PXF,Motorola 的 C-5 DCP 等;并行处理:每个内核都可以完成相似的任务,多个内核彼此间可并行执行,这种结构的网络处理器主要有 Intel 的 IXP,IBM 的 PowerNP 以及 Agere 的 PayloadPlus 等.

(2) 硬件多线程技术.为了进一步提高处理器的利用率,网络处理器通常引入硬件多线程(multi-thread)技术,且线程间的切换开销为 0.例如,Intel IXP2400 的每个微引擎中拥有 8 个线程,每个线程拥有独立的程序计数器 PC.IBM PowerNP 中也有类似的结构.

(3) 优化的内存管理和DMA单元.在一般的多处理器系统中,内存操作往往是系统的瓶颈.而路由器要对分组进行存储和复制等处理,需要执行大量的存储器操作.网络处理器为了优化这些操作引入经过特殊优化的存储器接口和DMA单元,以提高存储器的操作效率.例如,Intel IXP2400提供了SRAM队列以及相应的处理指令.

(4) 网络专用的协处理器.现在大部分网络处理器为一些特定的网络操作提供专用的硬件协处理[10],比如专用的路由表查找引擎、硬件分类引擎、缓冲和队列管理引擎.这些协处理器有些集成在网络处理器片内,有些则安排在片外.

综上可知,网络处理器是一种可编程的针对网络应用优化的多内核多硬件线程的处理器,通过良好的体系结构设计,网络处理器将软件的灵活性和硬件的高性能特点结合在一起.多数网络处理器还拥有协处理器,并对内存管理作了优化.要设计适用于网络处理器的算法或应用就必须考虑到网络处理器的这些特点,特别是多内核、多硬件线程(multi-core and multi-thread)和内存管理的特性.而我们提出的AM-Trie高速并行分组分类算法,正是基于网络处理器的这些特点而设计的.

2 AM-Trie分类算法描述

分组分类的主要性能指标包括:时间复杂度、空间复杂度、可扩展性、灵活性、最坏情况下的性能、更新复杂度以及预处理时间等,这些性能指标之间有些是互相冲突的(例如时间、空间复杂度),而且对于不同的应用这些指标的重要性也各不相同.目前,Internet的速度越来越高,服务种类日益增多.因此,为了适应Internet的发展趋势,分组分类算法应该优先考虑时间复杂度、可扩展性、灵活性这几个性能指标.

然而,传统的基于树形结构或者哈希的算法,查找的时间复杂度与分类字段的长度成正比.以二叉Trie树为例,对于IPv4的源目的地址就需要构建32层的二叉Trie树(对于哈希表而言,则是需要建立32个哈希表),IPv6的源目的地址更是需要构建128层的二叉Trie树,查找的时间代价太大了.为此,我们希望通过构建2^m叉Trie树来减少Trie树的层数,从而加快查找的速度.但是,由于目前Internet上广泛采用的CIDR技术,前缀的长度可以是任意介于0至该分类字段宽度之间的长度.这样的分类规则,数据类型并不能方便地转化成为多叉Trie树,因为这里采用2叉树(即1-bit trie)很好实现,而用多叉树则会遇到前缀长度不正好的情况(例如,我们采用4叉树,那么就是以2-bit为一个单位,遇到前缀长度为奇数的情况就无法表示了).

AM-Trie算法的第1个思想是:利用冗余来表示任意长度的前缀.考虑一个k-bit长的串,所有可能的前缀是$*,0*,1*,...,\overbrace{11...1}^{k-1}*,\overbrace{00...0}^{k},...,\overbrace{11...1}^{k}$共有$2^{k+1}-1$种,将它们顺序编号为$1,2,\ldots,2^{k+1}-1$,则它们可以用一个$(k+1)$-bit的串来表示.换算关系为:如果一个前缀为$P*$,$P$的长度为$p$-bit,那么它对应的"序号"为$2^p+P$.例如:*的前缀序号是1,1111*的前缀序号是31.利用这种"冗余"的表示方式,可以用一个$k+1$比特的串来表示所有k比特串的前缀.接下来,我们就可以利用这样的表示方式来构造合适的多叉Trie树——AM-Trie.

2.1 AM-Trie的构造和搜索

利用冗余表示前缀的方法,其好处在于它可以表示任意长度的前缀.因此,对于某一个字段的前缀,我们可以把它切割成任意的可变长(variable stride)的几段来构造AM-Trie树.假设我们已经确定了所要构造的AM-Trie树的层数和每一层的宽度,AM-Trie共有l层,每一层的宽度分别是$h_1,h_2,\ldots,h_l$比特,也就是说,AM-Trie树的每一层分别有2^{h_i}个节点,$\sum_{i=1}^{l} h_i = w$,w是这个分类字段的位宽.在创建某个规则表的AM-Trie树时,依次加入每一条规则.当加入一条规则时,首先将前缀按既定的方案分段;然后用上述冗余表示法依次计算每一段的序号,同时将指针指向对应层的该序号位置;当规则的所有段都处理完毕时,指针指向的位置即为规则要加入的位置.如果该位置已有规则存在(通常发生在更新规则库或者规则碰撞的情况),那么比较两个规则的优先级,将优先级高的加入即可;如果该位置没有规则,直接加入该条规则.之后继续加入下一条规则,直到处理完整个规则表为止.

我们用一个例子来直观地描述一下AM-Trie的创建.AM-Trie树中的每个节点包含pNext和pRule两个指

针,分别指向其子节点的起始位置和该节点所对应的规则,如果该节点没有子节点或者在规则表中没有直接对应的规则,那么对应的指针置为空指针.图 1 是根据表 1 生成的 AM-Trie,这里我们预先把规则分为两段,每一段都是 2-bits.

Table 1 A simple policy table
表 1 一个简单的规则表

Rule	Field
R0 (default)	*
R1	0*
R2	01*
R3	011*
R4	1111

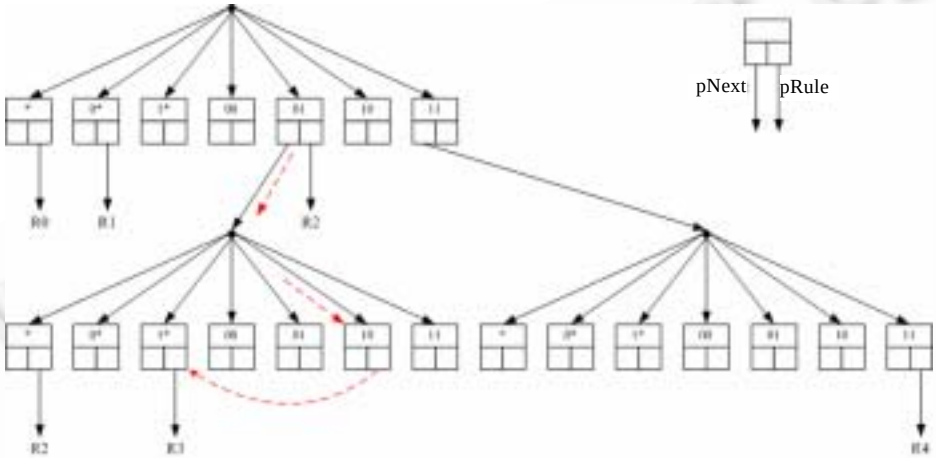

Fig.1 AM-Trie creation and search
图 1 AM-Trie 树的构造和搜索

首先加入第 1 个规则 R0,它只有一段(*),对应的序号是 1,于是把 R0 加入到根节点下的第 1 个子节点处.同样,R1,R2 被加入到根节点下的第 2 和第 5 个子节点处.规则 R3 有两段,第 1 段(01)的序号为 5,在 5 号子节点下面继续处理第 2 段(1*).因此,R3 被加入到 5 号子节点的第 2 层第 3 个子节点处.由于第 2 层的(*)位置及其的父节点表示同样的前缀(01*),也可将其父节点的规则 R2 复制到(*)处以便查找.同理,R4 也被加入到图示位置.如果还存在一条规则 R5 与 R3 一样都是 011*,那么,在加入 R5 时会发现它要加入的位置已经有一个规则 R3 了,这时需要比较 R3 和 R5 的优先级,以判断该位置放置哪条规则.

我们从生成的 AM-Trie 可以看出:在树的每一层中只有右边一半才有可能有子节点,这是因为前 $2^{h_i}-1$ 个节点对应的前缀以通配符'*'结尾,它们不可能有子节点.于是,这棵树看起来就左右不对称了,这也是非对称多权 Trie 树(asymmetrical multi-bit trie,简称 AM-Trie)这个名字的由来.

AM-Trie 的搜索与普通树的搜索类似,只是在达到叶节点后如果还没有查到对应规则的话,要回溯继续查找它的前缀节点.例如:到达分组的分类字段是 0110,先取它的第 1 段 01(序号 5),查找根节点下的第 5 个子节点.发现该子节点还有子节点,继续取分类字段的第 2 段 10(序号 6),查找该节点下的第 6 个子节点,发现它既没有子节点也没有直接对应的规则,则开始查它的前缀节点 1*(序号 3).发现 3 号节点下包含规则 R3,于是查找完毕并返回结果.查找过程的图示见图 1 虚线部分.

2.2 AM-Trie的压缩、加速和更新

从前一节的搜索过程中可以看到:当 AM-Trie 的叶节点不包含规则时需要回溯查找,导致搜索效率下降.仔细研究之后我们发现:当规则表确定之后,任何前缀所对应的含有规则的最长前缀也就确定了.因此,可以在预处理时就把每个节点的规则设为其最长前缀所对应的规则,这对于一个前缀来说实际上是一个"推(push)"的过

程,即把规则“推”到它覆盖(一个前缀和属于它的任何一个规则的关系称为覆盖,例如 1*覆盖了 10 和 11)的节点上去.如此进行下去,搜索到的叶结点就必然包含规则,不必回溯查找了.同时,AM-Trie 树的左边一半包含通配符‘*’的节点的信息都被推到右边被其覆盖的节点去了,在搜索时肯定不会被访问到,因此可以删除,这样就减少了约一半的存储空间.

图 2(虚线箭头代表“推”,虚线节点表示可以释放的节点)直观地显示了 AM-Trie 树的压缩过程和效果.压缩后的 AM-Trie 存储空间只有原来的一半,搜索时只要搜索到叶结点即可返回结果,不必回溯.最坏情况下,搜索次数等于树的高度,搜索的时间复杂度为 $O(h)$,h 为 AM-Trie 树的高度.

AM-Trie 树支持增量更新,增加操作在前面 AM-Tire 树的创建部分已经有所介绍.AM-Trie 的删除规则操作也很简单:先通过搜索操作定位到该规则的节点,然后从该节点中删除 pRule 指针即可;如果规则的删除导致这一层子树都不包含规则,那么删除这一层子树.在实际操作中,可以利用 0 号子节点来保存这一层包含规则的规则总数,以方便删除操作.AM-Trie 树的添加和删除操作的复杂度都是 $O(h)$,h 为 AM-Trie 树的高度.

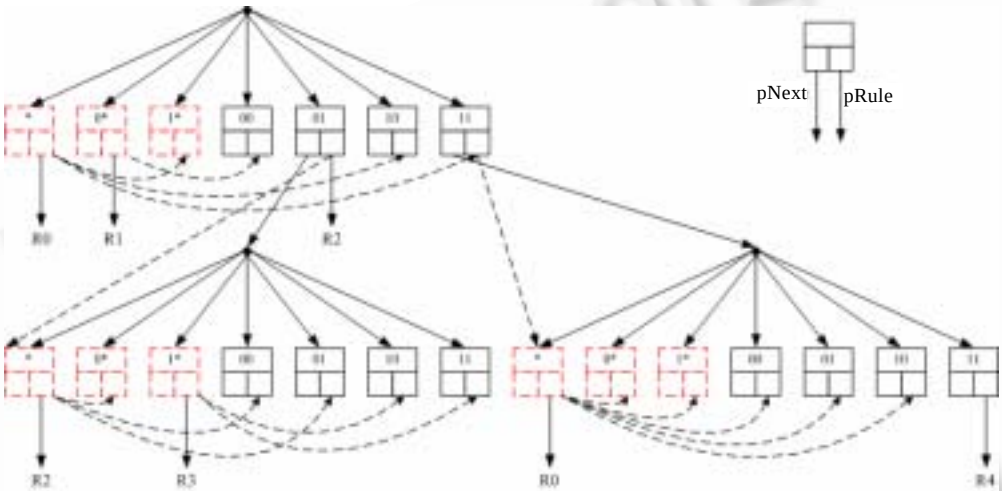

Fig.2 AM-Trie compression

图 2 AM-Trie 的压缩过程示意图

2.3 AM-Trie进行多维分类及算法复杂度分析

在进行多维分组分类时,我们借鉴了聚敛位向量(ABV)[2]的思想:对于一个 d 维分组分类问题,对分类规则的每个维度都建立一棵 AM-Trie 树,AM-Trie 树的叶结点存储的是利用 ABV 算法计算出来的位向量和聚敛位向量.由于各维度的搜索是独立的操作,因此可以并行地进行(在网络处理器上体现为多个微引擎或者硬件线程并行搜索).最后,将各维的查找结果进行 AND 操作得到最终的位向量,从而得出匹配所有维度的最终结果.

对于一个有 N 条规则的 d 维分类规则库,利用 AM-Trie 算法进行分类.假设建成的 d 棵 AM-Trie 树最高的一棵高度为 h(通常 $h<<W$),则有以下性质:

性质 **1**. AM-Trie 初始化复杂度为 $O(N\log_2 N)$.

创建每一棵 AM-Trie 树的时间复杂度是 $O(Nh)$,但在创建聚敛位向量(ABV)时,需要对规则库进行一次排序,故初始化复杂度为 $O(N\log_2 N)$.

性质 **2**. AM-Trie 搜索复杂度为 $O(h+d)$.

每棵 AM-Trie 树搜索得到 ABV 最坏情况下需要 h 次比较,不同 AM-Trie 树的搜索可以并行完成;从 ABV 得出最后前缀匹配的结果需要 d 次访存.故 AM-Trie 搜索复杂度为 $O(h+d)$.

性质 **3**. AM-Trie 算法空间复杂度为 $O(N^2)$.

共有 n 条规则,因此每个域的前缀数 $P\leq N$.如果某一层为 m-bit,结点如果展开下一层将产生 2^m 个子结点.在

最坏情况下(在最末一层才展开有效结点而且极为分散),需要的存储空间为

$$[(h-1)\times N+1]\times 2^m\times sizeof(Node)+N\times sizeof(ABV),$$

其中,$sizeof(Node)$是每一个节点的大小(8 个字节,两个指针);$sizeof(ABV)$是每个聚敛位向量的大小(N-bits).即 AM-Trie 算法的空间复杂度为 $O(N^2)$.

表 2 是 AM-Trie 算法与现有算法比较的结果.可以看出,AM-Trie 多维分类的优点是算法速度快,可以并行执行,同时又具有很好的可扩展性,可支持超大的分类规则库.而且,分类域或规则数越多,算法的优势就越明显.

Table 2 AM-Trie compared with current algorithms

表 2 AM-Trie 与现有算法的比较

Algorithm	Time complexity	Space complexity	Parallel execution
RFC[1]	$O(\log N)$	$O(N^d)$	No
ABV[2]	$O(W+d)$	$O(N^2)$	Yes
HyperCut[3]	$O(\log N)$	$O(N^d)$	No
AM-Trie	$O(h+d)$	$O(N^2)$	Yes

3 AM-Trie 每一层宽度的选择

在前一节的 AM-Trie 算法描述中,我们假定分类字段已被确定应该分成几段.但是,分类字段应该分成几段呢?如何划分才能兼顾时间复杂度和空间复杂度?这个问题,国际上已经有一些相关的研究工作[11].下面,我们从理论上进一步来分析这个问题.

在实际的规则表中,前缀的长度分布很不均匀,图 3 是我们随机选取的一个 BGP 表[12]中各种长度前缀分布的情况.该表中共有 80 194 条规则,可以看出:绝大多数规则都属于少数的几种长度,其中前缀长度为 24 的规则数更是超过了一半.一个最直接的想法就是:尽量选取出现概率大的前缀长度作为分段处,这样可以减少存储空间的开销.

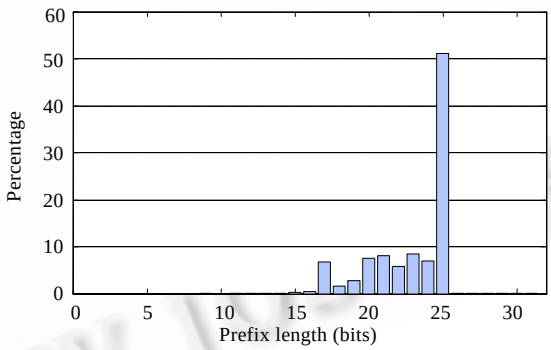

Fig.3 The prefix length distribution of a BGP table

图 3 某 BGP 表中的前缀分布情况

3.1 固定长AM-Trie

先考虑 AM-Trie 每一层内宽度相同(不同层之间可以不同)的简单情况,看看 AM-Trie 树的高度和存储空间的关系如何.我们用 $R[N,F_1(W_1),F_2(W_2),\ldots,F_d(W_d)]$来表示一个 d 维分类表,其中,N 为规则数;$F_i(W_i)$表示第 i 个分类字段有 W_i 比特.

引理 **1**. 对于一个分类字段 $F(W)$,如果其对应的 1-bit trie (即二叉 trie 树,binary trie)中的第 i 层和第 $i+j$ 层选作 multi-bit trie 相邻的两个切割点,那么,这一层的 multi-bit trie 带来的空间开销是$[nodes(i)-leaves(i)]\times 2^j$,其中,$nodes(i)$和 $leaves(i)$分别表示在 1-bit trie 中第 i 层的节点个数和叶节点个数.

证明:因为第 i 层的叶节点不必展开,其他每个节点展开到第 $i+j$ 层都扩展了 2^j 倍;而第 $i+1$ 层到第 $i+j$ 层的扩展都被第 i 层的扩展节点覆盖了.所以,这一层的 multi-bit trie 带来的空间开销是$[nodes(i)-leaves(i)]\times 2^j$. □

定理 **1**. 对于一个确定的分类表 R 其任一分类字段 $F_i(W_i)$以及给定的高度 $h\le W$,存在一个划分 $l_1,l_2,\ldots,l_h$ $(l_1+l_2+\ldots+l_h=W)$,使得生成的 h 层固定长 AM-Trie 树所占空间最小,且求解该优化问题的时间复杂度为 $O(h\times W^2)$.

证明:我们用 $S(x,y)$来表示把 x 层 1-bit trie 展开生成 y 层的 multi-bit trie 所带来的最小存储空间的开销.如图 4 所示,由一个 W 层的 1-bit trie 生成一个 h 层的 multi-bit trie 可以分两步完成:先由前 m 层的 1-bit trie 生成一个 $h-1$ 层的 multi-bit trie;然后,再把剩下的 $W-m$ 层生成第 h 层.生成 h 层的 multi-bit trie 的代价是这两步代价之和.

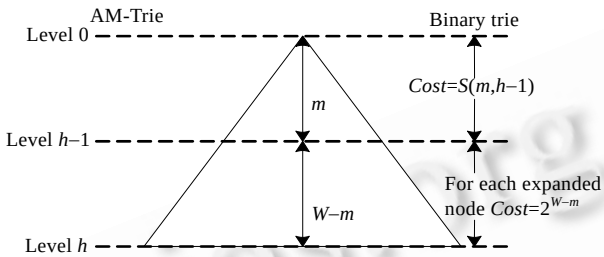

Fig.4 Recursion analysis of fixed stride AM-Trie

图 4 固定长 AM-Trie 递归分析示意图

根据引理 1,可以得到下面的关系:

$$S(W,h)=\min_{h-1\le m\le W-1}\{S(m,h-1)+[nodes(m)-leaves(m)]\times 2^{W-m}\} \tag{1}$$

以及初始条件

$$S(W,1)=2^W \tag{2}$$

这个问题可以用动态规划的方法来求解.代价函数 S 的两个变量的取值范围分别为 0~W 和 0~h,可转化为 $h\times W$ 个子问题,每个子问题涉及到一个求最小值的操作,这个优化过程的时间复杂度是 $O(h\times W^2)$. □

不难发现,式(1)递归展开之后,将会包含许多类似 $[nodes(m)-leaves(m)]\times 2^{W-m}$ 的项,其中 $leaves(m)$就是前缀长度为 m 的节点数;而$(W-m)$则是这一层的位宽.由此可以得到启发式的控制目标:为了使 AM-Trie 空间占用尽可能地小,在确定了 AM-Trie 树的层数之后,应该尽量从前缀长度比例大的层次进行分割.

3.2 可变长**AM-Trie**

AM-Trie 任意一个节点下的子树都可以有不同高度(位宽)的情况比较复杂,这时候,各条路径的长度也可能不同.但是,可变长情况下的最优化问题依然可解.

定理 **2**. 对于一个确定的分类表 $R[N,F_1(W_1),F_2(W_2),\ldots,F_d(W_d)]$的任一分类字段 $F_i(W_i)$,以及给定的高度 $h\le W$,存在一系列划分 $l_{n1},l_{n2},\ldots,l_{nk}$ $(l_{n1}+l_{n2}+\ldots+l_{nk}=W,\ 1\le n\le N,\ 1\le k\le h)$,使得生成的高度不超过 h 的可变长 AM-Trie 树所占空间最小,且求解该优化问题的时间复杂度为 $O(N\times h\times W^2)$.

证明:用 $C(M,r)$表示从节点 M 开始,把它和它的子孙节点扩展成至多 r 层的变长 AM-Trie 树所需要的开销.定义 $height(M)$为节点 M 到它的任意一个子孙节点的最长路径的长度.$height(M)=0$ 表示这个结点是叶结点.另外,扩展 $nodes(s)$函数为 $nodes(M,s)$,用来表示原 1-bit trie 第 s 层上 M 的子孙节点的个数.

如图 5 所示,把一棵 1-bit Trie 树扩展成高度不超过 h 的变长 AM-Trie 树,可先从根节点 G 生成 AM-Trie 树的第 1 层,再以这一层的子节点生成高度不超过 $h-1$ 的变长 AM-Trie 树,由此可得出下面的递归关系式:

$$C(M,h)=\min_{1\le m\le 1+height(G)}\left\{2^m+\sum_{M'\in Nodes(G,m+1)}C(M',h-1)\right\} \tag{3}$$

以及初始条件

$$C(M,1)=2^{1+height(M)} \tag{4}$$

这个问题同样可以用动态规划的方法求解,与固定长情况相比,可变长 AM-Trie 的优化问题还需要对子节

点的开销求和,因此时间复杂度是 $O(N\times W^2\times k)$. □

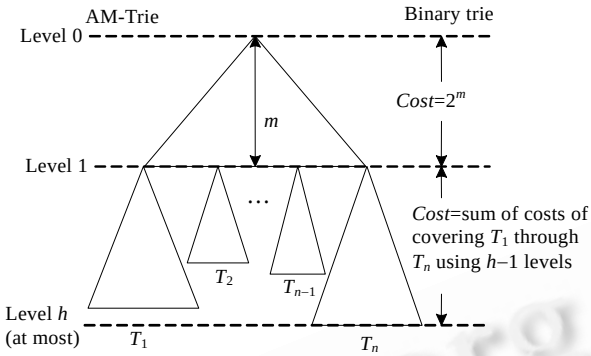

Fig.5 Recursion analysis of variable stride AM-Trie

图 5 可变长 Trie 递归分析示意图

4 性能测量与分析

我们在 Intel IXP2400 网络处理器上实现了 AM-Trie 算法,8 个微引擎分配如下:以太网帧的接收、发送、分组调度以及缓冲管理各用 1 个微引擎实现,余下的 4 个微引擎并行进行分组分类以及 IP 协议的处理.

在进行实测时,我们根据如图 3 所示的前缀分布情况随机生成规则库,然后利用 IXIA 1600 流量发生器向 IXP2400 中输入分组,分组的大小均为 64 字节.系统处理的分类字段是 TCP/IP 六元组(源 IP、目的 IP、源端口、目的端口、TOS 域和协议类型),所有的 6 个字段都支持最长前缀匹配.在实现中,6 棵 AM-Trie 树最高为 3 层,根据前面的时间复杂度公式,处理一个分组只需要 15 次访存.即便是用 150MHz 这样速度较低的 DDR SDRAM,理论上 AM-Trie 也可达到 10Mpps 的线速.

图 6 为规则数从 32~4096 情况下的系统吞吐率,可以看出,整个系统的吞吐率并不随着规则库的增大而降低,始终保持在 2.5Gbps(5Mpps)以上.这有两个原因:其一是 AM-Trie 算法的可扩展性很好(利用层级式结构,AM-Trie 可以很容易地扩展到支持 256K 条规则),算法性能受规则数影响很小;其二是 IXP2400 是针对 OC48 (2.5Gbps)的网络应用设计的,AM-Trie 算法已经最大限度地利用了 IXP2400 的处理能力,受到系统其他模块(例如接收、发送和调度等)的制约,整个系统的性能无法再继续提高.

图 7 展示了 AM-Trie 算法存储开销和规则数之间的关系(为了简化实现,我们把算法中的位向量长度都固定为 4096 比特,因此,内存开销呈线性增长,而不是理论上的 $O(N^2)$空间复杂度).相对于硬件分组分类算法,AM-Trie 属于“用空间换时间”的算法,利用速度不高但容量大且价格便宜的 RAM 来达到与昂贵的硬件(FPGA,TCAM 等)相当的性能.IXP2400 支持 2×64M SRAM 以及 2G DRAM,对 AM-Trie 算法而言绰绰有余.

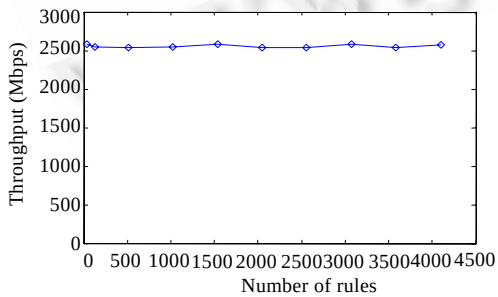

Fig.6 The throughput of AM-Trie on IXP2400

图 6 基于 IXP2400 的 AM-Trie 算法的吞吐率

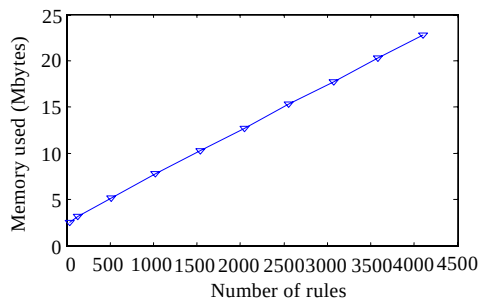

Fig.7 The memory cost of AM-Trie on IXP2400

图 7 基于 IXP2400 的 AM-Trie 算法的存储开销

我们在 Intel 的高端网络处理器 IXP2800 上初步实现了该算法,经测试表明,系统吞吐率接近 10Gbps.代码

和系统还在进一步的优化之中,预期可以达到 10Gbps(即 20M pps)的吞吐率.

5 结束语

Internet 的飞速发展需要高速的分组分类作基础.当前的高速分组分类通常用成本非常高且发热量大的硬件(如TCAM和FPGA)实现.为了解决这个问题,本文针对当前高速网络应用对分组分类算法的要求以及网络处理器体系结构的特点,提出了一种高速多维分组分类算法——AM-Trie (非对称多权 Trie 树)算法.该算法创造性地利用冗余表示任意长度的前缀,降低了 Trie 树搜索的复杂度;通过压缩 AM-Trie 树来消除回溯,进一步提高了搜索速度并减少了存储空间.AM-Trie 算法可并行执行,很好地利用了网络处理器多内核、多硬件线程的特点;算法的可扩展性良好,规则数的增加对算法性能几乎没有影响.AM-Trie 算法的时间复杂度为 $O(\log W)$,W 为分类字段的比特数,空间复杂度为 $O(N^2)$,N 为分类规则数.此外,本文还从理论上给出了一种启发式算法来设计 AM-Trie 树的每一层的宽度,在确定 AM-Trie 树层数(时间复杂度)的情况下达到了存储空间(空间复杂度)最小.

在此基础上,我们基于 Intel 网络处理器 IXP2400 (最高速率为 OC48)实现了该算法,性能分析和测量结果表明,AM-Trie 算法在 TCP/IP 六元组分类中性能达到了 2.5Gbps 的线速(分组大小均为 64 字节,亦即 5Mpps),并且在更高性能的网络处理器(例如 IXP2800)上还可以有很大的性能提升空间.同时,我们的实验结果也指出,“空间换时间”是网络处理器上利用廉价的 RAM 存储器获得高速分类性能的一条可行之路.

References:

[1] Gupta P, McKeown N. Packet classification on multiple fields. In: Proc. of the ACM SIGCOMM 1999. 1999. 147−160. http://tiny-tera.stanford.edu/~nickm/papers/Sigcomm99.pdf

[2] Baboescu F, Varghese G. Scalable packet classification. In: Proc. of the ACM SIGCOMM 2001. 2001. 199−210. http://www.cs.ucsd.edu/groups/sysnet/miscpapers/p2-baboescu.pdf

[3] Singh S, Baboescu F, Varghese G, Wang J. Packet classification using multidimensional cutting. In: Proc. of the ACM SIGCOMM 2003. 2003. 213−224. http://www.sigcomm.org/sigcomm2003/papers/p213-singh.pdf

[4] Baboescu SSF, Varghese G. Packet classification for core routers: Is there an alternative to cams? In: Proc. of the IEEE INFOCOM 2003. 2003. 53−63.

[5] Lakshminarayanan ARK, Venkatachary S. Algorithms for advanced packet classification with ternary cams. In: Proc. of the ACM SIGCOMM 2005. 2005. 193−204.

[6] Network processing forum (npf). http://www.npforum.org/

[7] Network systems design conference. http://www.networkprocessors.com/

[8] Wolf T, Franklin MS. Design tradeoffs for embedded network processors. In: Proc. of the ARCS 2002. 2002. 149−164.

[9] Shah N. Understanding network processors. Technical Report, 2001. http://www.gigascale.org/pubs/338.html

[10] McAuley AJ, Francis P. Fast routing table lookup using CAMs. In: INFOCOM (3). 1993. 1382−1391.

[11] Srinivasan V, Varghese G. Faster IP lookups using controlled prefix expansion. In: Measurement and Modeling of Computer Systems. 1998. 1−10.

[12] BGP routing table analysis reports. http://bgp.potaroo.net/

郑波(1978－),男,博士生,主要研究领域为分网络处理器,网络传输控制,系统性能评价,无线网络安全.

林闯(1948－),男,教授,博士生导师,CCF高级会员,主要研究领域为系统性能评价,计算机网络,随机 Petri 网,逻辑推理模型.

曲扬(1978－),男,博士生,主要研究领域为系统性能评价,网格计算,工作流模型.