一种采用按值分支树的多维流分类算法 [1]

姚兴苗 胡光岷 李乐民

(电子科技大学宽带光纤传输与通信系统技术国家重点实验 成都 610054)

摘 要: 该文针对 Modular 算法用于流分类所存在的问题，提出一种采用按值分支树的多维流分类算法．算法支持对规则维数和数量的扩展，并能同时处理前缀匹配和范围匹配．仿真试验结果表明：该算法具有良好的扩展性，支持大容量的分类规则．

关键词: 按值分支树，多维流分类，范围匹配，前缀匹配

中图分类号: TN919.2 **文献标识码**: A **文章编号**: 1009-5896(2004)09-1413-07

A Multi-dimensional Packet Classification Algorithm with Trees Divided by Value

Yao Xing-miao Hu Guang-min Li Le-min

(*Nat. Key Lab of Broadband Opt. Fiber Transm. and Comm. Networks, UEST of China, Chengdu* 610054, *China*)

Abstract In order to solve problems of modular algorithm, a multi-dimensional packet classification algorithm that uses trees divided by value is presented in this paper. It supports increment of rule's dimension and scale. Moreover, it can deal with prefix match and range match. The simulation result shows that the algorithm is scalable and practical for large-scale rules.

Key words Trees divided by value, Multi-dimensional packet classification, Range match, Prefix match

1 引言

随着因特网的不断发展，因特网服务提供商希望网络能提供更好的服务，例如能提供服务质量 (QoS) 保障，提供防火墙等，为此路由器必须具有流分类能力．经过人们的研究，提出了很多种流分类算法，以下是较为典型的几种流分类算法。

使用 CAM(Content Addressable Memory)[1] 的流分类方法，它具有快速的查找时间 $O(1)$ 和合理的空间 $O(N)$(N 为规则数量) ．但是它存在以下的缺点：集成度低，功耗高，可扩展性差，具有特定的位宽并且位宽有限，不直接支持范围匹配。文献 [2, 3] 提出了两种改进的算法，改进的算法虽然在一定程度上减少了使用的 CAM 位宽或者容量，但还是会受到 CAM 其他方面缺陷的制约。

重复流分类 (Recursive Flow Classification, RFC)[4] 算法是一种多维的流分类算法，其优点是查找时间快并能同时支持前缀和范围匹配，但即使针对较少的规则数量，需要的存储空间也很大，因此算法不易扩展。

位向量 (Bit Vector, BV)[5] 算法将流分类问题看作经典计算几何问题，算法便于硬件实现，在小规则数量时具有快速的查找速度．但随着规则数量的增加，查找的时间线性增加，所需存储容量也很大。聚合位向量 (Aggregated Bit Vector, ABV)[6] 算法使用聚合位向量的方法改进了 BV 算法的查找时间，但是所需要的存储容量略有增加。

[1] 2003-04-19 收到， 2003-10-28 改回

元组空间搜索 (Tuple Space Search, TSS)[7] 算法把具有相同前缀长度组合的规则组织成一个多元组，并对每一个元组使用 Hash 函数进行规则匹配。文献 [8] 提出的算法对 TSS 算法进行了改进，节省了查找时间。但是由于 Hash 函数具有不确定性，使得基于 Hash 函数的算法在发生冲突时效率很低。

随着路由器对流分类的要求愈来愈高，流分类规则数量和规则维数不断扩展，上述的算法并不能完全满足路由器要求。为此流分类算法应该综合使用多种方法，按流分类过程本身的特点，将其划分为几个阶段，每个阶段根据自身的特点和流分类的总体目标进行不同的处理。Modular 算法 [9] 实际上就是这样的一类算法，使用三个阶段的查找方法： (1) 索引跳转表 (Index jump table) ：所有的规则依据规则串中的某些比特位组成索引将整个规则集合划分为不同的较小的规则集合。 (2) 查找树：每个索引跳转表表项所对应的规则集合组成一棵 2^m 叉的查找树。当查找树中某节点剩下的规则数目小于某个给定的值时，分类中止于叶子节点，即叶子规则束 (Filter bucket) 。 (3) 叶子规则束：算法对叶子规则束包含的规则不做进一步的区分。

Modular 算法将每一维规则看作前缀的形式，使它能够适应规则的扩展，在规则数量方面，作者使用了一个 1 兆条规则的分类器，仍然取得了较为满意的流分类速度。然而它只支持前缀匹配，不能够直接支持范围匹配。而在多维的规则匹配中，有些维是需要范围匹配的。要匹配这些规则，只有将范围匹配转化为前缀匹配的形式，实际上算法支持的规则将成倍地减少。由于 Modular 方法使用索引跳转表，某些规则可能在子树和叶子规则束中重复多次，这样可能会引起存储空间的爆炸。针对算法存在的这些问题，我们提出了一种适用于多维、大容量、可扩展的按值分支的高效流分类查找算法，并和 Modular 算法以及其他算法进行了比较。

2 采用按值分支树的多维流分类算法

我们的算法采用三阶段的查找方法。第一阶段使用索引跳转表 (简称索引表) 和索引规则束 (Index bucket) ，第二阶段使用按值分支树和树根规则束 (Root bucket) ，第三阶段查找叶子规则束。首先来说明我们的按值分支树算法。

2.1 基本算法——按值分支树

流分类在需要范围匹配时变得比前缀匹配更加复杂。文献 [1] 指出，可以把范围转化成前缀的形式，然后使用最长前缀匹配的算法来处理。设流分类规则的维数为 d ，每维的宽度为 W 比特。对于每维 W 比特宽度的范围，至多可转化为 $2W-2$ 个前缀；最坏情况下，对于一个 d 维的规则，转化后的前缀数量将达到 $(2W-2)^d$ 。现有的算法中，文献 [10] 提出了一种算法将范围转化为前缀而规则数量并没有多大的扩展，但算法只是针对一些特定的多维分类规则。大多数的其他算法都不能支持直接的范围匹配，本身这些算法支持前缀的数量就不多，如果是间接支持范围匹配，那么支持的规则数量就更加有限。我们提出的算法将有效地解决这个问题，算法无需对规则范围转化为前缀而是直接处理，如果规则是前缀的形式，仍然将它作为范围进行处理。

为了说明按值分支树算法，我们用选用流分类器中的一些规则作为例子。表 1 中如规则 R_1 ，源 IP 地址 202.115.8.0/24 表示规则为前缀的形式， 24 为前缀的长度；协议 * 表示通配符，即规则 R_1 对分组的目的端口不进行过滤。针对表 1 所示的规则集合，首先将规则集合写为范围的形式，见图 1 。

图 2 中，按值分支树的根节点 0 包含所有的规则 $\{R_1,R_2,R_3,R_4,R_5,R_6\}$ ，如果我们选取其中一个规则某一维范围的端点值 (约定为左端点) 来区分这些规则，规则将被这个端点值区分到根节点的左右子节点。如我们选择规则 R_1 第一维的左端点值 V_1 来区分，左子节点 1 包含的规则为 $\{R_4,R_5,R_6\}$ ，右子节 2 点包含的规则为 $\{R_1,R_2,R_3\}$ ；对于左子节点 1 ，选择规则 R_6 第二维的左端点值 V_6 来区分，同理规则将被区分到下一级子节点。但是如果我们对根结点 0 选择 R_4 第四维的 V_4 来区分，左右子节点包含的规则分别为 $\{R_1,R_2,R_3,R_5,R_6\}$,

表 1　流分类器中的一些规则

规则	源 IP 地址	目的 IP 地址	源端口	目的端口	协议
R_1	202.115.8.0/24	202.120.13.0/24	23	*	TCP
R_2	202.115.8.23/28	202.120.13.21/28	1024	gt 1023	UDP
R_3	202.115.8.23/32	202.120.28.21/32	gt 1023	gt 1023	UDP
R_4	202.115.3.69/32	202.118.3.35/25	23	1024	*
R_5	202.115.3.69/28	202.118.3.35/28	gt 1023	23	TCP
R_6	202.115.3.69/28	202.118.22.69/32	1024	gt 1023	TCP

$\{R_1, R_2, R_3, R_4, R_6\}$；如果对根结点 0 选择 R_3 第二维的 V_3 来区分，左右子节点包含的规则分别为 $\{R_1, R_2, R_4, R_5, R_6\}$, $\{R_3\}$。相对来说，这两个端点值选择得不好：前者使得子节点规则重复过多，增加存储容量；后者使得子节点的规则数量不平衡，会导致生成树的深度较深。因此，端点值的选择关系到生成树的性能。于是我们期望对每个节点都选用一个最优的值 (使区分后的子节点规则重复最少，左右节点平衡) 来区分此节点对应的规则，使生成树具有很好的性能，虽然这不能保证按此过程生成的树为最优，但我们期望它是具有合理深度的平衡分支树，最后的仿真算法验证我们的这个想法。

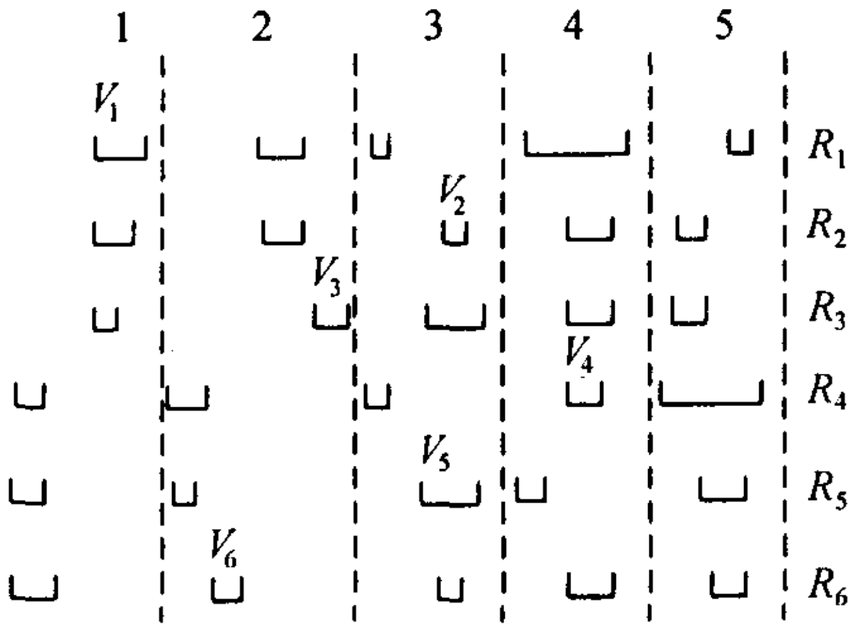


图 1　将规则写成范围的形式

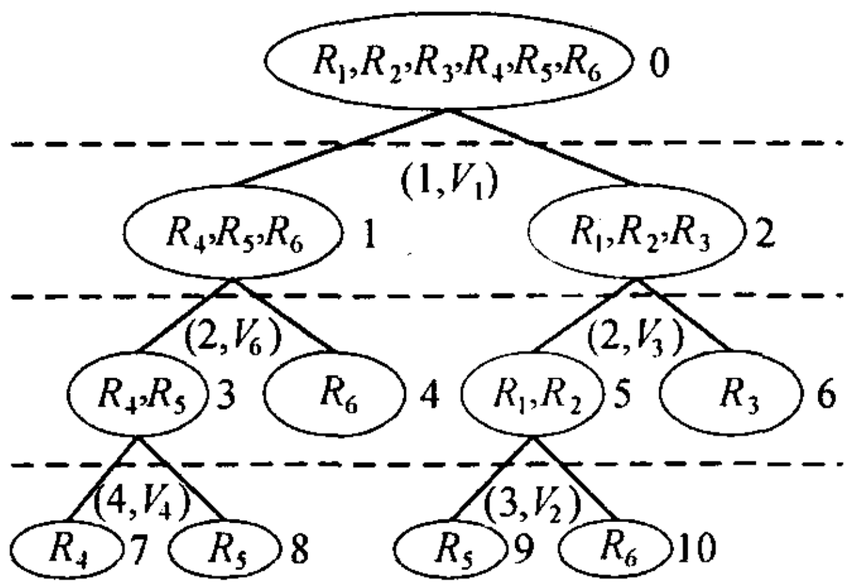


图 2　表 1 所示的规则的按值分支树生成

以下是我们选取端点值算法的详细描述：

设 RT_{sn} 是二叉树节点 sn 所对应的规则的集合 (sn 为节点的序号)，N 是根结点所对应的规则的个数 (也是所有规则的统一序号的最大值)，K 是规则对应的维数。按值分支树节点 sn 按值 (j, V_i) 分支，$j \in [0, K], V_i$: $i \in [0, N]$，表示节点 sn 选取节点本身对应的某一规则 R_i(统一序号为 i 的规则) 的第 j 维左端点值 V_i 来区分 sn 节点对应的集合。

$L_j^{V_i}(\mathrm{RT}_{sn})$ 表示按值 (j, V_i) 将 sn 节点对应的规则区分到左子节点的个数，$R_j^{V_i}(\mathrm{RT}_{sn})$ 表示按 (j, V_i) 区分到右子节点的个数，$\mathrm{LR}_j^{V_i}(\mathrm{RT}_{sn})$ 表示按 (j, V_i) 区分到左右子节点的规则个数 (即规则重复的个数)。同时，我们定义：

$$D_j^{V_i}(\mathrm{RT}_{sn}) = |R_j^{V_i}(\mathrm{RT}_{sn}) - L_j^{V_i}(\mathrm{RT}_{sn})|$$

$D_j^{V_i}(\mathrm{RT}_{sn})$ 表示区分到左右子节点的规则的数目的差。

$D_{\max}(\mathrm{RT}_{sn})$, $D_{\min}(\mathrm{RT}_{sn})$ 分别表示在 $(j \in [0, K], V_i : i \in [0, N])$ 情况下，$D_j^{V_i}(\mathrm{RT}_{sn})$ 的最大和最小值；$\mathrm{LR}_{\max}(\mathrm{RT}_{sn})$, $\mathrm{LR}_{\min}(\mathrm{RT}_{sn})$ 分别表示在 $(j \in [0, K], V_i : i \in [0, N])$ 情况下，$\mathrm{LR}_j^{V_i}(\mathrm{RT}_{sn})$ 的最大和最小值。

设按值分支树 sn 节点的分支平衡因子为 $\mathrm{DB}_{sn}=\alpha(D_{\max}(\mathrm{RT}_{sn})-D_{\min}(\mathrm{RT}_{sn}))$，分支树 sn 节点的分支重复因子为 $\mathrm{LRB}_{sn}=\beta(\mathrm{LR}_{\max}(\mathrm{RT}_{sn})-\mathrm{LR}_{\min}(\mathrm{RT}_{sn}))$，$\alpha$, β 分别为平衡系数和重复系数.

我们定义参考值:

$$V_j^{V_i}(\mathrm{RT}_{sn})=\frac{D_j^{V_i}(\mathrm{RT}_{sn})}{\mathrm{DB}_{sn}}+\frac{\mathrm{LR}_j^{V_i}(\mathrm{RT}_{sn})}{\mathrm{LRB}_{sn}}$$

使 $V_j^{V_i}(\mathrm{RT}_{sn})$ 的取值为最小值 $V_{\min}(\mathrm{RT}_{sn})$ 的 (j,V_i) 即为 sn 节点所选的端点值.

重复迭代对按值分支树的每一个节点使用上述的按值分支算法，直到叶子节点包含的规则数小于某一给定的值，就可以得到一个较为平衡合理的分支树. 按照上述算法，表 1 所示的流分类规则的按值分支树生成如图 2 所示，此时取 $\alpha=\beta=0.5$，叶子规则束包含的规则数量为 1 .

2.2 采用按值分支树的多维流分类算法

在第一阶段查找中，由于索引表的选取，同一规则可能被复制多次，这可能引起空间爆炸，是一个隐藏的危机. 以一个 2 维的规则为例子: $R=(10**,0****)$，若取规则每维的前 3 bit 作为索引，规则将被复制 2^3 次. 因此如果取索引的比特为 m 位，对应一个规则的这 m 位包含 $*$ 的个数为 n，每一规则将被复制 2^n，我们称 n 为复制因子. 为了避免空间过度的膨胀，在我们的算法使用索引规则束来避免规则的复制: 对于某一个规则，如果 n 大于我们设定的某一个值 n_t，规则不进入索引表，添加到索引规则束中.

对于第二阶段的查找，Modular 算法检测冲突的规则是不够的，只考虑到规则完全覆盖的情况. 以两个 2 维的规则例子来说明: $R_1=(1011*,001**)$, $R_2=(101**,0****)$, R_2 完全覆盖 R_1，可以用消冗余的方式消除 R_1；如果 $R_1=(1011*,0****)$, $R_2=(101**,001**)$, R_2 第一维覆盖 R_1, R_1 第二维覆盖 R_2；不能消除. R_1 和 R_2 可能在建树的时候被重复多次，为了消除这个潜在的问题，相冲突的规则不到树中扩散，而是添加到树根规则束中. 如果相冲突的规则太多，实际上这样的数据库是不安全的，可以在树根规则束采用文献 [11] 解决冲突检测的算法来实现. 第三阶段，查找叶子规则束中包含的规则.

上述的三个阶段分别使用了索引规则束、树根规则束和叶子规则束. 这些规则束构成的多级规则束解决了规则多次复制，节省了存储空间.

以下是我们采用按值分支树的多维流分类算法的详细描述，算法分为初始化过程和查找过程. 初始化过程的具体描述如下:

步骤 1 初始化索引规则束，置索引规则束的规则个数为零，Num(Index_Bucket)=0；初始化 2^m 个索引表表项 $\mathrm{IT}(2^m)$，每一索引表表项对应查找树的根节点；初始化每个索引表表项对应的树根规则束，置树根规则束的规则个数为零，$\mathrm{Num}(\mathrm{IT}(2^m).\mathrm{Root\_Bucket})=0$.

步骤 2 遍历数据库中规则，取每维的前缀构成 m 比特索引值，如果复制因子 $n\geq n_t$，规则不进入索引，添加到索引规则束，Num(Index_Bucket)++；如果复制因子 $n<n_t$，添加到相对应的索引表表项中.

步骤 3 遍历每一个索引表表项，检测相冲突的规则，将冲突规则添加到树根规则束，$\mathrm{Num}(\mathrm{IT}(2^m).\mathrm{Root\_Bucket})$ 增加；对剩余的无冲突规则，建立如 2.1 节描述的按值分支树，直到叶子规则束的个数小于某个设定的值为止.

算法的查找过程为

步骤 1 取分组头相应每维的比特构成 m 比特，查找到对应的索引表 $\mathrm{IT}(2^m)$.

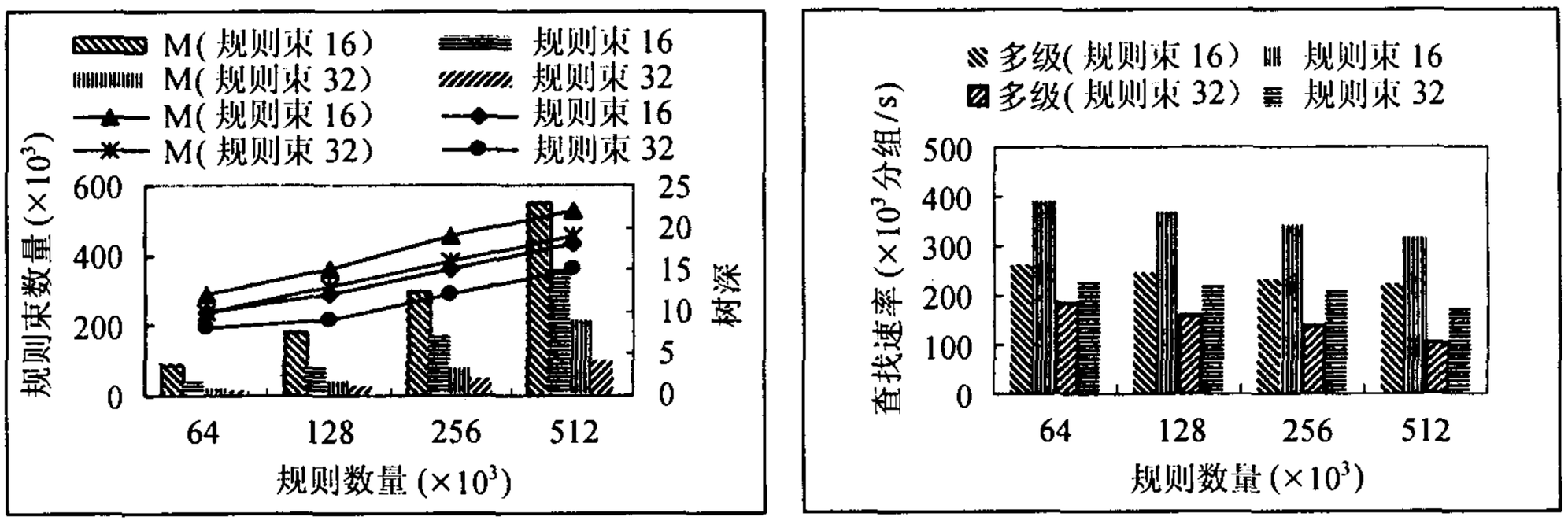
步骤 2 查找 $\mathrm{IT}(2^m)$ 对应的按值分支树，直到返回 $\mathrm{IT}(2^m).\mathrm{Leaf\_Bucket}$；

步骤 3 依次查找索引规则束 Index_Bucket，返回的树根规则束 $\mathrm{IT}(2^m).\mathrm{Root\_Bucket}$ 和叶子规则束 $\mathrm{IT}(2^m).\mathrm{Leaf\_Bucket}$，得到相对应的规则.

3 仿真结果和讨论

实际的流分类数据库很多是通过人为配置的，并且规则的数量十分有限，为了对算法进行评测，我们生成了一个模拟的流分类数据库。

图 3 是我们的算法与 Modular 算法的比较 (叶子规则束的取值为 16 和 32，复制因子 $n_t = 6$)。在这组数据的比较中，本文算法没有使用多级规则束方法。仿真的结果与我们的预想基本一致，在算法的空间上，由于本文算法不需要将范围匹配拆分为前缀匹配，规则束的个数明显少于 Modular，而叶子规则束也终止在树较浅的深度。随着叶子规则束可包含的规则增多，我们发现对于两种算法，规则束个数和树的深度的差距减小，实际上是因为规则束可包含的规则增多阻止了规则的扩散，规则重复的次数减少。在算法的时间方面，同样是本文算法优于 Modular 算法。



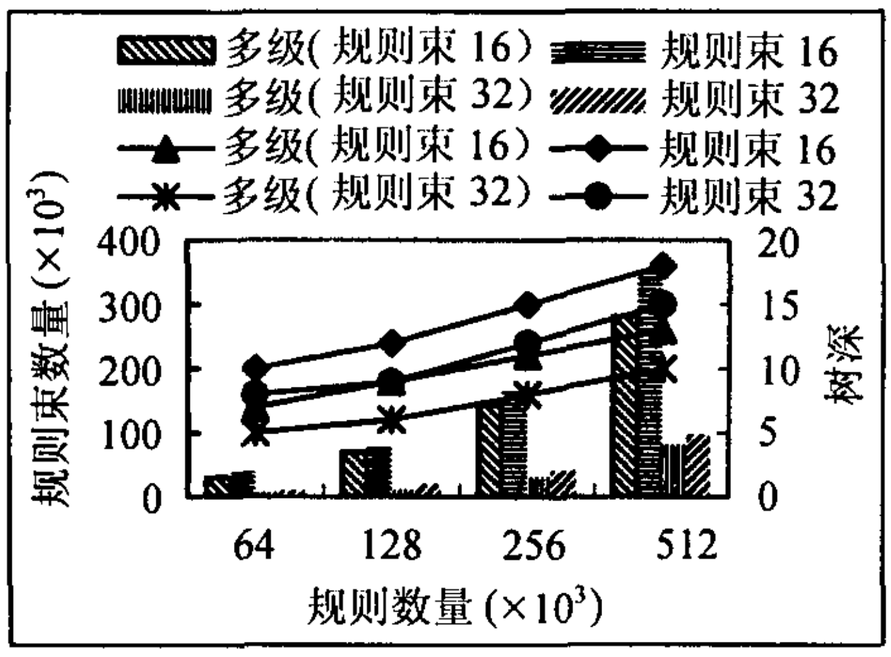
(a) 树的统计　　(b) 查找速率

图 3　本文算法与 Modular 算法的比较

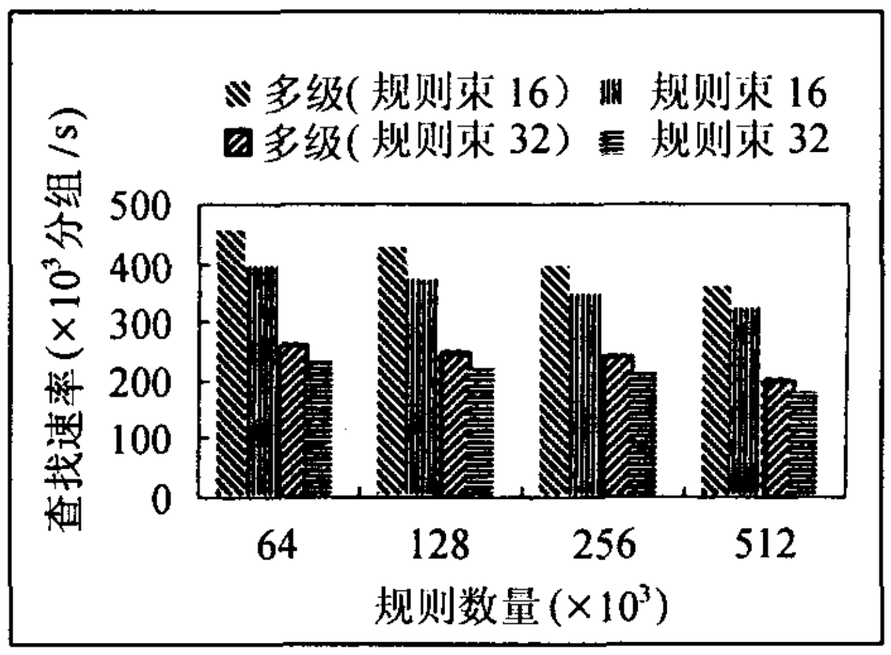
图 4 是本文算法在有多级规则束与无多级规则束下的一个比较，在所需的空间上，前者的规则束个数少于后者。在仿真时，我们发现如果多级规则束包含的规则太多，查找时间会增加很多，主要的原因是我们的多级规则束中的规则都采用线性查找。解决的办法是只有某一规则扩展得很厉害时，我们才将它放入索引规则束中，以减少线性查找的规则的数量；或者对于规则束中的规则使用某些特别的查找方案，如采用查找树、硬件并行查找的方式等。图 4 (b) 的结果就是在索引规则束也使用 2.1 节所述的查找树算法所得的结果。结果表明，使用多级规则束可以有效地避免规则不必要的复制，能保证空间不无限制地膨胀；如对多级规则束采用简单的线性查找降低了算法的时间性能，那么采用另外特别的查找方法将消除这一不好的影响。

下面我们将评测本文算法与各种算法不在同规则数量下的相对性能 (查找时间为每匹配一个分组所需的时间)，如表 2— 表 4 所示。由于一些算法只能适应一定数量的规则 (当规则的数量超出一定的数量，算法的时间或者需要的存储空间变得不可接受)，我们在不同的规则数量下评价的算法并不完全相同。

从时间的性能方面来看，RFC 算法 (采用三阶段的缩减树) 在规则数量较少时的性能最好；BV 算法随着规则数量的增加，时间性能变得极差，查找时间几乎是线性的增长； ABV 算法 (比特聚合值 $A = 32$) 改进了 BV 算法的查找时间，但与 Modular 算法与本文算法相比，查找时间受规则数量的变化影响较大。



(a) 树的统计



(b) 查找速率

图 4 本文算法比较

表 2 几种典型算法的相对性能比较结果 (较少规则数量)

	1×10^3		2×10^3	
算法	查找时间 (s)	所需的存储器容量 (Mbyte)	查找时间 (s)	所需的存储器容量 (Mbyte)
RFC	0.0000002	16.81	0.0000003	128.75
BV	0.0000067	0.26	0.0000152	0.82
ABV	0.0000039	0.27	0.0000062	0.84
Modular	0.0000021	0.35	0.0000026	0.51
本文算法	0.0000014	0.21	0.0000015	0.38

表 3 几种典型算法的相对性能比较结果 (较多规则数量)

	16×10^3		32×10^3	
算法	查找时间 (s)	所需的存储器容量 (Mbyte)	查找时间 (s)	所需的存储器容量 (Mbyte)
BV	0.0000716	31.81	0.0001629	68.39
ABV	0.0000125	32.80	0.0000173	70.53
Modular	0.0000031	3.82	0.0000035	7.23
本文算法	0.0000020	2.12	0.0000023	3.53

表 4 几种典型算法的相对性能比较结果 (大容量规则)

	256×10^3		512×10^3	
算法	查找时间 (s)	所需的存储器容量 (Mbyte)	查找时间 (s)	所需的存储器容量 (Mbyte)
Modular	0.0000043	48.92	0.0000048	116.56
本文算法	0.0000025	19.42	0.0000028	36.18

从各种算法消耗的存储器空间来看， RFC 算法消耗的空间随着规则数量的增加而急剧增加，不能适应较多规则数量的流分类查找；与本文算法相比较， BV 算法和 ABV 算法所需要的空间受规则数量的影响也很大。

综合来看，几种算法都能适应较少的规则数量，尤其是 RFC 算法在此情况下的时间性能最好，但是所需的存储容量太大，随着规则数量的增加， RFC 所需要的储存空间将变得不切实际。而本文算法随着规则数量的增加，与 RFC 算法、 BV 算法、 ABV 算法以及 Modular 算法相比，查找时间和所需要的存储器空间变化平稳，因此综合性能是最好的。

4 结束语

流分类问题是宽带通信网络中的关键技术之一，是 IP QoS 、防火墙等问题的基础，近年来得到了广泛的研究。我们针对 Modular 算法存在的问题，提出了一种易扩展的按值分支树的多维流分类算法。与几种典型的流分类算法相比较，本文算法同时支持前缀匹配和范围匹配，能处理大型的流分类数据库；由于对于规则每维的匹配类型没有严格的要求，规则的维数也容易扩展。本算法不仅可以软件实现，还容易实现硬件的查找，是一种值得应用的流分类算法。

参　考　文　献

[1] Gupta P, McKeown N. Algorithms for packet classification. *IEEE Network*, 2001, 15(2): 24–32.

[2] van Lunteren J, Engbersen T. Fast and scalable packet classification. *IEEE J. on SAC.*, 2003, 21(4): 560–571.

[3] Liu Huan. Efficient mapping of range classifier into ternary-CAM. Proceedings of 10th Symposium on High Performance Interconnects, Stanfoord, California, 21-23 Aug. 2002: 95–100.

[4] Pankaj P, McKeown N. Packet classification on multiple fields. Proceedings of ACM, Cambridge, MA, USA, September 1999: 147–160.

[5] Lakshman T V, Stiliadis D. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. Proceedings of ACM Sigcomm, Vancouver, Canada, September 1998: 191–202.

[6] Baboescu F, Varghese G. Scalable packet classification. Proceedings of ACM Sigcomm, San Diego, Califonia, Aug. 2001: 199–210.

[7] Srinivasan V, Suri S, Varghese G. Packet classification using tuple space search. Proceedings of ACM Sigcomm, Cambridge, MA, USA, September 1999: 135–146.

[8] Wang Pi-Chung, Chan Chia-Tai, *et al.*. Fast packet classification through tuple reduction and lookahead caching. ICON 2002 10th IEEE International Conference on Networks, Singapore, Aug. 2002: 197–202.

[9] Thomas Y C Woo. A modular approach to packet classification: Algorithms and results. Proceedings of IEEE Infocom, Tel Aviv, Israel, March 2000, vol.3: 1213–1222.

[10] Feldmann A, Muthukrishnan S. Tradeoffs for packet classification. Proceedings of IEEE Infocom, Tel Aviv, Israel, March 2000, vol.3: 1193–1202.

[11] Hari A, Suri S, Parulkar G. Detecting and resolving packet filter conflicts. Proceedings of IEEE Infocom, Tel Aviv, Israel, March 2000, vol.3: 1203–1212.

姚兴苗：　男，1976 年生，博士生，目前主要研究方向为高速路由器的流分类和路由查找.

胡光岷：　男，1965 年生，博士，教授，目前主要研究方向为通信网与宽带通信技术.

李乐民：　男，1932 年生，教授，博士生导师，中国工程院院士，目前主要研究方向为通信网与宽带通信技术.