IP 分类技术研究

喻中超,吴建平,徐 恪
(清华大学计算机科学与技术系,北京 100084)

摘 要: 网络应用的发展要求路由器必须有能力支持防火墙、QoS、流量计费等一系列功能. 这就要求路由器对 IP 包进行分类,根据分类结果完成对数据包的不同处理. 本文全面地介绍了 IP 分类技术研究的最新成果,以及 IP 分类的典型算法. 最后本文对其中三种典型算法在虚拟环境下做了评测,比较了它们的优缺点.

关键词: IP 分类; 查找算法
中图分类号: TP393 **文献标识码**: A **文章编号**: 0372-2112 (2001) 02-0260-03

Study of IP Classification Technology

YU Zhong-chao, WU Jian-ping, XU Ke
(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084, *China*)

Abstract: The development of Internet applications necessitates routers' ability to support those functions such as firewalls, provision of QoS or traffic billing etc. All these functions need classification of packets, based on which it is determined how different packets are processed subsequently. In this article, the authors survey the recent advances in the research of IP classification and introduce some of the typical algorithms. The authors also evaluate three of them in a virtual environment and compare their pros and cons.

Key words: IP classification; lookup algorithm

1 引言

一些网络应用希望路由器能够提供相应的功能支持. 这些功能包括防火墙,基于策略的路由(policy-based routing)、区分服务(differentiated qualities of service)、QoS、流量计费等等. 它们都是以包分类技术为基础的. 由于路由器处理的数据包绝大部分都是 IP 包,因此相应的技术也称为 IP 分类(IP classification)技术.

2 IP 分类问题

考虑具有 N 条过滤规则的过滤规则数据库 F_{dat},给定一个包头 H,在 F_{dat}中可能有不止一条过滤规则与 H 匹配. 我们约定,与每条过滤规则 F 相联系有一个代价函数 *cost*,过滤规则 F 的代价记为 $cost(F)$. 定义最佳过滤规则匹配问题为在 F_{dat}中查找满足下列条件的过滤规则 f_{best}: f_{best}是与 H 匹配的过滤规则;在 F_{dat}中不存在其它的过滤规则 F, F 与 H 匹配并且满足

$$cost(F) < cost(f_{bast})$$

IP 分类问题是最佳过滤规则匹配问题的一个实例,其核心是高效的查找算法. 一般来说,一个好的查找算法,必须满足以下三个条件: (1) 速度快; (2) 占用内存少; (3) 易于更新. 目前的大多数解决方案都比较重视前两个因素,即查找的时间效率和空间效率,而这两个因素往往互相制约.

3 几个典型 IP 分类算法

3.1 RFC 算法

RFC(Recursive Flow Classification[1])算法是一种多维 IP 分类快速查找算法. 其主要思想是将 IP 分类问题看成一个将包头中的 S 比特数据到 T 比特的 classID 的一个映射($T = \log N$ 且 $T \ll S$, N 是过滤规则的总数). 如果预先计算出包头中的这 S 位共 2^S 种不同情况中每种情况所对应的 classID 值,那么每一个包只需要一次查表,但是这样会消耗极大的空间. RFC 的思想是通过多步或者说是多个阶段(phase)完成这个映射过程. 每个阶段的结果将一个较大的集合映射成一个较小的集合,称其为一次缩减(reduction).

图 1 中是 4 阶段的缩减树,对 IP 包查找的过程如下:

(1) 在第一阶段(Phase 0),包头中的 K 个域被分成若干个块(chunks),每一块被用来作为并行查找的索引; (2) 在后面的几个阶段,每次查找内存的索引值都是由前几个阶段的查找的结果通过特定的方式合并而成的; (3) 在最后一个阶段,查找的结果得到一个确定的值,这个值就是该包对应的 classID.

RFC 算法的性能受到两个参数的影响: (1) 选取的阶段(Phase)的数目 P; (2) 在给定 P 的情况下,"缩减"树的形状,也就是后面的阶段的索引值从前面哪几个阶段的查找结果进行合并. 在 P 和缩减树固定的情况下,随着过滤规则树 N 的增加,消耗的内存量也增加. 对于同样的过滤规则数据库,一

收稿日期:2000-04-03;修回日期:2000-08-26
基金项目:国家自然科学基金(No. 6982202 和 No. 69822003);国家"八六三"重点攻关项目(No. 863-306-ZD-07-01)

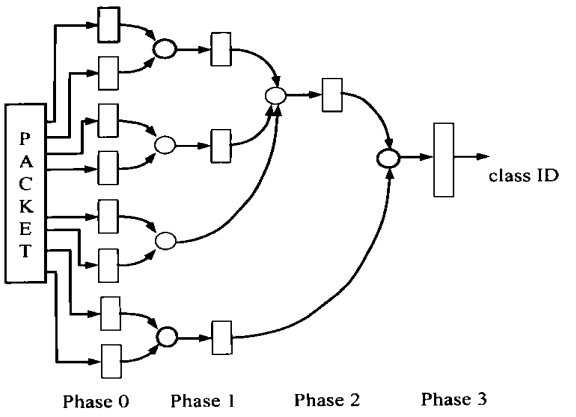

图 1　一个四阶段的 RFC 缩减树

般而言 P 越大消耗的空间越少,但是查找时间变慢.

RFC 查找的时间复杂性是 $O(1)$. 但是消耗内存非常大,有时候甚至是不切实际的.

3.2　Modular 算法

准确的讲,Modular 算法[2]应当算作一类算法. 它假定与每个过滤规则 F_i 相联系有一个权值 W_i. 在知道了 W_i 的情况下,Modular 算法试图建立更加有效的查找数据结构. Modular 算法将查找过程分为三个层次.

(1) 跳转表 ——所有过滤规则依据规则位串中的某些位划分为不同的组,该位串通常是过滤规则不同域的前缀,其选择通常也是根据统计特性作出;

(2) 查找树 ——(1)中同组的过滤规则组成一棵 2^m 叉查找树. 通过每次检查过滤规则中的 m 位,将其分为 2^m 组. 这 m 位是从当前规则位串中尚未检查的各位中选择. 其选择通常遵循两个原则:减少重复拷贝(节省空间)和 2^m 棵子树的平衡(节省平均查找时间和最坏情况查找时间). 建立查找树的过程实际上就是一个不断分组的过程,分组中止于叶子节点,即 filter bucket;

(3) Filter bucket ——当剩下的过滤规则数目小于某个给定的值,则不再做进一步的划分,该节点即为叶子节点. 由于叶子节点中的过滤规则数目比较少,因此可以采用与(2)中利用查找树查找不同的更为简单的查找方法.

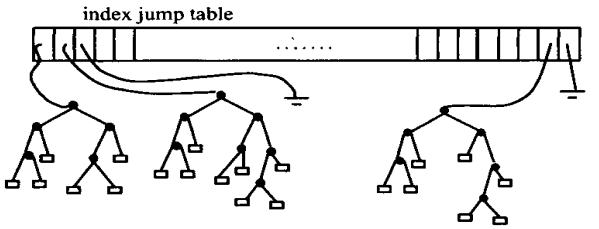

图 2　Modular 算法查找数据结构示意图

查找过程如下. 整个包头中的全部域视为一个比特串,称为包头位串. 首先以包头位串中的特定位为索引查跳转表,得到查找树的入口地址;然后,沿着查找树,每次根据包头位串中 m 位的值不断"下降",直到到达叶子节点. 然后在叶子节点中,查找最小代价的过滤规则. Filter bucket 是最终存储过滤规则的地方,对 filter bucket 中过滤规则的查找,可以有不同的方法,例如可以用硬件流水线来进行线性查找.

Modular 的时间复杂性和空间复杂性比较复杂. 在极坏的情况下时间和空间都可能达到无穷(不收敛).

3.3　Cross-Product 算法

由于 IP 包头的分布不具有局部性,所以对包头进行 Cache 的效率不高. Cross-Product[3]的 Cache 策略是基于对过滤规则中各域值的交叉组合(即 Cross-Product)的"缓存",每一个交叉组合对应一类 IP 包,因此用少量的 Cross-Product 就可以代表相当数量的一类 IP 包. 在 Cross-Product 中表现出的局部性也相应好得多,Cache 的效率也高得多.

在极端情况下,Cross-Product 可能会产生内存爆炸问题. 解决办法是,并不将所有的 cross-product 的最佳匹配过滤规则都存起来,而是固定实际存储的最佳匹配过滤规则的数量(类似硬件 Cache,称其为软 Cache),软 Cache 中的内容是动态更新的. 当一个包头 H 到达的时候,先对其按照各自的域得到最精确匹配,然后将不同域的查找结果连接起来,进行 Hash 查找. 在绝大部分情况下,都能够成功(称为 cache hit). 在不成功的情况下,说明相应的 cross-product 尚不在 cache 中,此时需要进行计算,得到最佳匹配前缀的 cross-product. 同时,根据某种置换策略将软 Cache 中的一个不常用的 cross-product 换出,换入当前的 cross-product. 该算法对过滤规则的形式有一定的限制.

3.4　元组空间查找算法

元组空间查找算法(Tuple Space Search[4])的出发点是,尽管过滤规则数据库中含有的不同前缀或范围(如端口范围)的数量很多,但是不同的前缀长度的数量往往是很小的. 因此,各种不同长度的交叉组合的数量也是很小的. 基于这种现象,定义一个不同域"长度"的组合为一个元组,所有元组的集合称为元组空间. 由于一个元组相应于每个域的比特位的情况是已知的,可以将这些比特连接(concatenation)起来组成一个 Hash 表的索引值,通过该索引值找到与该元组对应的过滤规则集合.

当包头 H 到达的时候,对每一个元组,查哈希表得到该元组中匹配的过滤规则,通过元组空间中的这种线性查找得到代价最小的过滤规则. 由于元组的数量很小,因此即使采用线性查找也是可行的.

3.5　无冲突哈希查找算法

无冲突哈希查找是我们提出的一种综合效率较高的查找算法.

过滤规则基本上只限于 5 个域:目的/源 IP、目的/源端口,协议域. 实际上,目的端口、源端口和协议域的取值往往是 0 - 65535 中极少的一部分. 目前协议域实际上能够取的值只能是 TCP、UDP、ICMP、IGMP、(E) IGRP、GRE 和 IPINIP. 在 Client-Server 结构中,服务器端口集中于少数几个值,极少过滤规则会对客户端端口产生兴趣,一般是要求其大于 1023[5]. 因此实际的过滤规则端口和协议域的取值(或取值范围)的不同情况的组合数目是非常有限的. 基于此点,建立了一个二阶段的查找表,可以进行无冲突的哈希查找.

无冲突哈希查找算法的基本思想是先假定过滤规则只有

源端口、目的端口和协议域,对包进行分类,称为三域等价类.根据上面的分析,三域等价类的数目应当是非常少的,以至于可以通过预先计算的方式算出所有包对应的三域等价类.为节省空间,通过两个阶段的查表完成这一过程.第一阶段分别以目的端口、源端口和协议域对数据包分别进行分类,得到三个 ID 值,分别记为 d, s 和 p,定义哈希函数 $g(d,s,p) = PSd + Ps + p$,其中 D、S 和 P 分别是以目的端口、源端口和协议域对包进行分类的类别总数.可以证明哈希函数 g 是无冲突的,也即如 $g(d_1,s_1,p_1) = g(d_2,s_2,p_2)$,则必有 $d_1 = d_2, s_1 = s_2, p_1 = p_2$.证明从略.

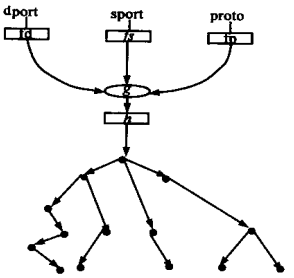

图 3 无冲突哈希 Trie 树算法

每个三域等价类对应一个目的/源 IP 前缀队的集合,当每个包找到自己对应的三域等价类之后,再根据自己的目的/源 IP 对查找最终的过滤规则 ID,这实际上是一个二维的 IP 分类问题.对二维 IP 分类,已经有比较成熟的算法,如 Grid-of-Tries[3],采用 Grid-of-Tries 的无冲突哈希多域 IP 分类算法见图 3 所示.

4 评测及结论

对三个典型算法在一个虚拟环境下进行了评测.实际中的过滤规则数据库往往是通过人为配置的,有较大的随机性,但是也有一定的规律性,过滤规则数据库的建模本身就是一个很大的题目.由于我们的目的是比较三者的相对性能,所以对模拟的要求不是特别严格.

表 1 三种典型 IP 分类算法性能相对比较结果

	# of Rule = 500			# of Rules = 600		
	Setup time(s)	Searching time per packet(s)	Max memory (MB)	Setup time(s)	Searching time per packet(s)	Max memory (MB)
RFC	40	0.0000005	28.413	79	0.0000006	42.349
Norr-col	0	0.0000039	1.721	0	0.0000039	1.760
Modular	0	0.0000136	0.400	0	0.0000138	0.434
	# of Rule = 700			# of Rules = 800		
	Setup time(s)	Searching time per packet(s)	Max memory (MB)	Setup time(s)	Searching time per packet(s)	Max memory (MB)
RFC	138	0.0000006	58.176	173	0.0000006	61.016
Norr-col	0	0.0000039	1.759	0	0.0000039	1.842
Modular	0	0.0000142	0.466	0	0.0000144	0.503
	# of Rule = 900			# of Rules = 1000		
	Setup time(s)	Searching time per packet(s)	Max memory (MB)	Setup time(s)	Searching time per packet(s)	Max memory (MB)
RFC	346	0.0000006	100.555	521	0.0000007	119.316
Norr-col	0	0.0000039	1.871	0	0.0000039	1.903
Modular	0	0.0000146	0.558	0	0.0000148	0.573

总共测量了 6 组数据,分别测量了三种算法的初始数据结构建立时间,包分类平均查找时间和消耗的最大内存.我们看到,大量的包的分类结果都是与 default 过滤规则对应的 classID.这种现象是可以理解的,认为它对应了一个主干路由器上的情况,因为主干网路由器的过滤规则的数目相对流过它的包来说是很小的,因此大部分的数据包的分类结果都是 default.

表 1 中显示的结果和预想的情况基本是吻合的.RFC 在时间效率上是最好的,按照假设,它总共需要 10 次查表(采用 3 阶段的缩减树,如果并行查找的话速度将会更快,相当于三次查表的时间)就可以得到 classID,且不受规则数目的影响.无冲突哈希查找(表中简称为 Norr-col)的查找时间基本上也不受过滤规则数目的影响,因为无冲突哈希查找算法只受 IP 地址最大宽度的限制,和过滤规则的数目并没有很直接的关系.从时间性能上看 Modular 的性能比较差,这和测量结果大都是 default 规则有关,它意味着大多数查找都要中止于查找树的某个深度较大的地方.

从空间效率上看,RFC 随着过滤规则的增加消耗的内存量也急剧增加,这从数据结构的建立时间也可以看出.Modular 算法的空间性能是最好的,一方面是我们主干路由器的假设尽可能避免了过滤规则的重复存储,另一方面 filter bucket 的存在防止了过滤规则的进一步扩散(在 filter bucket 中采用的是线性查找,这也是它时间性能较差的一个原因,实际中可以用更好的办法,如硬件流水线).

从综合性能来看,无冲突哈希查找是最好的.它的时间性能接近 RFC 算法,而空间消耗远比 RFC 小得多.更重要的是,它消耗的时间和空间受过滤规则的影响不是很大,因此其行为具有可预测性.我们提出的无冲突哈希 Trie 分类算法已经成功地用于国家"八六三"重点攻关项目"高性能安全路由器"的研制过程中.

参考文献:

[1] P. Gupta and N. Mc Keown. Packet classification on multiple fields [J]. ACM Computer Review, 1999, 29(4): 146 - 160.

[2] T. Y. C. Woo. A modular approach to packet classification: algorithms and results [J]. In: Gruein R ed. Proceedings of IEEE Infocom2000. San Francisco, CA: IEEE Computer Society Press, 2000: 1210 - 1217.

[3] V. Srinivasn, G. Varghese, S. Suri, et al. Fast scalable level four switching [J]. ACM Computer Review, 1998, 28(4): 191 - 205.

[4] V. Srinivasan, S. Suri and G. Varghese. Packet classification using tuple space search [A]. In Proceedings of ACM Sigcomm99 [C], Aug, 3 1999: 135 - 1466.

[5] W. R. Stevens. UNIX Networking Programming vol 1 (2nd Edition) [M]. Englewood Cliffs, NJ: Prentice Hall, Inc. 1998

作者简介:

喻中超 1977 年生,硕士研究生,研究方向是分组分类与调度算法.

吴建平 1953 年生,获博士学位,教授,博士生导师,研究方向是计算机网络体系结构,计算机网络协议测试,形式化技术.

徐 恪 1974 年生,博士研究生,研究方向是计算机网络体系结构,计算机系统性能评价.