支持快速增量更新的包分类算法①

周天贵　程海鹏　华　蓓 (中国科学技术大学 计算机科学与技术学院 安徽 合肥 230027)

摘　要： 动态数据包分类是目前新兴网络服务的基础，但现有包分类算法的更新性能不能令人满意。基于递归空间分解和解释器方法，设计和实现了一个支持快速增量更新的两阶段多维包分类算法 TICS，利用局部数据结构重建替换方法允许规则集增量更新，并通过适当的内存管理允许查找和更新的并行同步进行。实验表明，算法的更新速度比目前更新最快的 BRPS 算法至少提升了一个数量级，且内存消耗少，具有良好的并行扩放性。

关键词： 包分类；增量更新；并行

A Packet Classification Algorithm with Fast Incremental Update Support

ZHOU Tian-Gui, CHENG Hai-Peng, HUA Bei

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)

Abstract: Dynamic packet classification is the basis of emerging network services, but the update performance of existing packet classification algorithms is unsatisfactory. Based on the Recursive Space Decomposition and Interpreter approach, this paper designs and implements a two-stage multi-dimensional algorithm TICS with fast incremental update support. It allows incremental update of rule set by reconstructing and replacing the local data structure, and allows parallel synchronous execution of search and update through appropriate memory management. The experimental results show that TICS is at least an order of magnitude faster than the current fastest algorithm BRPS, with less memory consumption and good parallel scalability.

Keywords: packet classification; incremental update; parallel

1　引言

包分类是支撑各种高级网络服务的基础技术，如基于策略路由、服务质量、流量计费、防火墙等。包分类是根据预设的分类规则集将到达的数据包指派给不同数据流的过程。规则集是一系列规则的集合，每条规则包含与数据包头中的某些域对应的匹配条件、数据流标识和相应的处理方法。当数据包头中的相关域满足某规则的所有匹配条件时，称该数据包匹配该规则。目前典型的规则要求匹配包头中的源/目的 IP 地址、源/目的端口号和协议，称五维包分类。

一般从三个方面评价包分类算法的性能：分类速度，内存消耗，更新速度。目前大多数包分类算法主要关注提高分类速度和减小内存消耗，不太重视更新性能，添加或删除规则的操作很复杂，有的甚至需要重建整个数据结构。这使得这些算法对于要求动态分类的应用支持较差。

支持增量更新的包分类算法主要有递归空间分解[1]、EGT[2]、HiCuts[2]、HyperCuts[2]和 BRPS[3]等。递归空间分解是基于决策树的二维前缀分类算法，每条规则只保存在一个结点中，更新代价最小，但树高度较大。EGT、HiCuts 和 HyperCuts 都是基于决策树的多维分类算法，由于一条规则可能存放于多层树结点中，更新代

① 收稿时间:2009-06-11

价较高。BRPS 采用分层的范围和前缀扩展列表二分查找，时空性能较前面算法有较大提升，更新速度是目前多维包分类算法中最快的，但每次更新仍需约 10 万个时钟周期，规则的存储也有冗余。

本文结合递归空间分解和 TIC[4]的解释器方法，设计和实现了一个支持快速增量更新的两阶段五维包分类算法 TICS (TIC with Space-decomposition)。该算法利用递归空间分解实现 TIC 的第一阶段以支持增量更新，通过路径压缩技术改进递归空间分解的查找性能，并通过数据结构局部重建替换和适当的存储管理措施允许更新和查找并行进行，避免了因更新导致的查找停顿。实验表明，TICS 更新速度比 BRPS 至少提升了一个数量级，并且 TICS 的并行扩放性很好，可以通过提高算法的并行度来提高分类速度。

本文剩余内容安排如下：第 2 节简要介绍与本文算法直接相关的已有工作；第 3 节介绍 TICS 算法的设计思想及关键技术；第 4 节给出实验结果并分析；第 5 节总结全文。

2 相关工作

本节简要介绍递归空间分解算法和 TIC 算法。

在递归空间分解[1]中，地址长度为 w 比特的所有地址对(S, D)构成二维空间中一个 $2^w \times 2^w$ 的正方形区域。用一对地址前缀表示的规则 R = (S*, D*) 对应该区域中一个 $2^{w-i} \times 2^{w-j}$ 的矩形，其中 i、j 分别为 S*、D*的前缀长度。在图 1 的示例中，假定 IP 地址长度 w = 6，规则 R8 = (010*, *) 对应图 1 中一个 $2^3 \times 2^6$ 的矩形。一个数据包 P 对应二维地址空间中的一个点，若该点被规则 R 的区域所覆盖，称数据包 P 匹配规则 R。若一个点被多个规则的区域所覆盖，则包分类最终要确定包含该点的优先级最高的区域(最佳规则)。

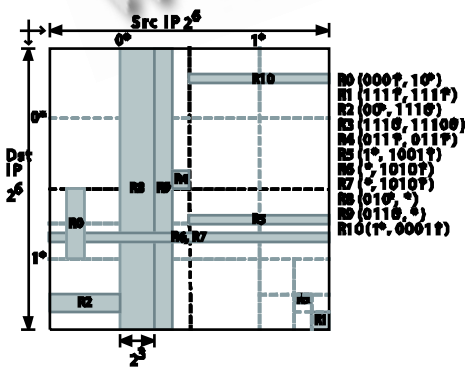

图 1 规则 R0~R10 对应的区域

四叉树用来表示地址空间在两个维度上的递归划分。从根结点(对应整个二维地址空间)开始，每一层上，一个正方形区域被划分成四个相同大小的子正方形。树中每个结点对应一个正方形，它的 4 个子结点分别对应由该正方形进一步分解得到的四个子正方形。空间分解树建立了空间区域与树结点的对应关系。

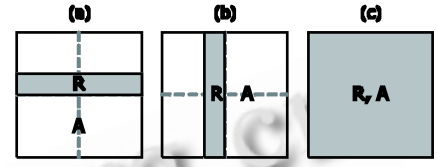

图 2 规则 R 映射到区域 A 的三种情形

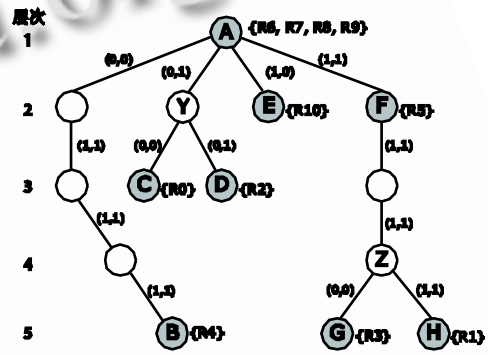

图 3 根据规则 R0~R10 构造的决策树

各分类规则按照以下原则映射(保存)到树中的一个结点：如果结点 v 对应的区域 A 是包含 R 的最小正方形(见图 2)，则将规则 R 映射到结点 v。所有规则映射完毕后，停止空间分解过程。图 3 是根据图 1 的规则集构造的决策树，带阴影的结点保存了分类规则，其它为空结点。

TIC[4]是基于解释器的两阶段五维包分类算法。第一阶段提取数据包中的源/目的 IP 地址，利用二维 RFC[2]得到匹配这对地址的所有候选规则。这组规则其余三个域的匹配条件在预处理阶段被编码成 CISC 风格的指令，并被组织在一个代码块中，第一阶段得到的其实是指向该代码块的指针。第二阶段利用解释器解释执行代码块，确定匹配规则。与五维 RFC 比，TIC 的空间复杂度和预处理时间都大大降低了。

3 TICS算法设计

3.1 问题分析

TIC[4]虽然保持了与 RFC[2]相当的分类速度，但因其第一阶段用了 RFC 而不支持增量更新。递归空间分

解[1]是更新局部性最好的算法，利用递归空间分解替换 RFC 来改进 TIC 的更新性能是一个很自然的想法。但是空间分解树高度较大，最坏情况下树高度达 33，影响查找速度。其次，与文献[1]中仅查找包含数据包 P 的最佳区域不同，这里需要找出包含 P 的所有区域，以便继续匹配其余三个域。

更新涉及对分类数据结构的修改。更新与查找的同步问题处理不好会影响查找性能，特别是那些需要频繁更新规则集的应用，如流量工程、基于入侵检测的联动防御等。

本文提出对 TIC 的一种改进算法 TICS，引入路径压缩技术，压缩空结点，改进查找性能，并采用局部数据结构重建替换和显式内存管理来实现更新与查找的并行同步进行。

3.2 路径压缩

空结点可被压缩，当且仅当：1)该空结点是其父结点的惟一孩子，或者 2)该空结点只有一个孩子。因部分路径被压缩，每个结点中需增加两个变量：变量 np(Node Prefix)保存与该结点代表的正方形区域对应的 IP 前缀对，用于判断到达的数据包是否匹配该结点；变量 bp(Bit Position)用于指示下一步分支选择使用 IP 地址中哪个位置的比特。

对于决策树，每个结点的 np 为该结点对应的前缀对，bp 为结点前缀长度加 1。路径压缩大致过程如下：1)先序遍历决策树，直至遇到空结点 node；2)若 node 是其父结点 parent 的惟一孩子，将 node 的所有孩子设为 parent 的孩子，并将 parent 的 bp 加 1；3)若 node 只有一个孩子 child，将 parent 中指向 node 的指针改为指向 child；4)继续遍历决策树。图 4 是图 3 决策树的路径压缩结果。

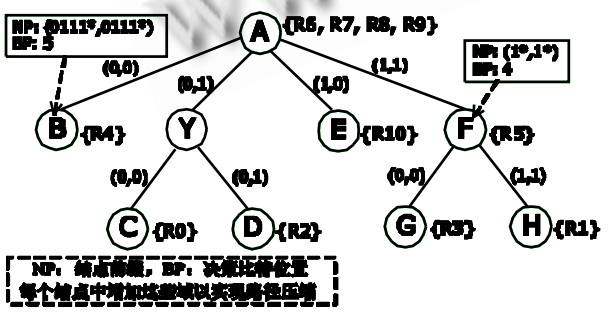

图 4　路径压缩后的四叉决策树

使用 Classbench[5]生成的规则集进行的实验表明，采用路径压缩后平均路径长度从29.1减小至9.8，压缩了约 2/3。

3.3 查找

根据图 2 的三种情形，将结点包含的规则集分为三个规则子表。其中在情形(a)和(b)对应的两个规则子表中，分别只需在 DstIP 和 SrcIP 方向上判断数据包 P 是否落入某条规则对应的区域。每条规则的非跨越维区间可用一对 IP 端点地址表示，这些 IP 端点地址将 DstIP 或 SrcIP 轴划分成一系列不相交的地址区间。每个地址区间可能对应了若干规则，将这些规则分别保存到对应的 IP 端点地址中，每个端点中规则的其余三个域的匹配条件在预处理阶段被编码到一个代码块中。情形(c)对应的规则都是该结点的候选规则，可直接将这些规则的其余三个域编码成代码块。

给定一个数据包 P，TICS 算法的查找过程为：从根结点开始，1)先判断 P 是否匹配当前结点前缀 np；不匹配则终止查找，目前记录的最高优先级规则即为匹配规则；否则，2)若结点非空，利用二分法查找结点内的 DstIP 和 SrcIP 方向上的两个 IP 端点列表，得到与 P 的源/目的 IP 地址匹配的候选规则代码块，加上情形(c)对应的代码块，利用解释器根据 P 的其余三个域解释查找得到的三个代码块，获得匹配的规则，记录到目前为止优先级最高的匹配规则；3)分别取 P 的源地址和目的地址的第 bp 比特组成 2 比特的分支选择关键字，进入到相应的孩子结点，返回 1)。

3.4 增量更新

当需要插入或删除一条规则 R 时，先根据 R 的 IP 前缀对查找决策树，找到插入或删除的位置。从对树结构的影响来看，插入和删除操作只可能出现两种情形：1)只需修改一个已有结点的内部查找结构，而不影响树结构；2)需要插入或删除一个或几个结点，从而影响到局部树结构。

本文算法在多核平台上运行时，一个更新线程负责维护规则集及分类数据结构，若干个包处理线程分类收到的数据包。为减小更新数据结构对查找性能的影响，本文采用局部树结构重建替换的方法，即先建立需要替换的局部树结构，然后修改相应树结点的指针指向新的局部结构。

图 5 为向图 4 的树中插入新规则 R11 的示例，其中浅灰色结点是新增结点，无箭头虚线是新建立的指针。插入过程大致为：根据规则 R11 的前缀对(111*,

111*)找到插入位置；构建 R11 所属结点 Z 及其内部结构,令结点 Z 的两个孩子指针分别指向结点 G 和 H；构建结点 F 的替换结点 F′ ,令 F′ 的孩子指针指向 Z；修改结点 A 的第四个孩子指针指向结点 F′ ，回收结点 F 的资源。整个插入过程只在最后一步修改了原树结构中结点 A 的一个指针。按照 Intel 的系统编程手册[6]：读写一个按照自身大小对齐的 1、2、4、8 字节的指针是原子的，因此该指针的修改在多核平台中是原子的。

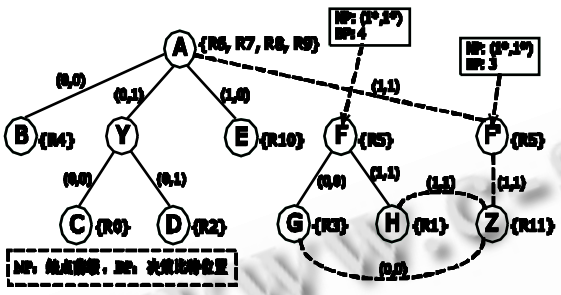

图 5　插入结点 Z 后的路径压缩四叉决策树

然而，在新的局部结构接入前后可能产生数据结构不一致。结点 F′ 接入决策树之后，访问 F′ 的所有查找线程将使用新规则集，而之前已进入结点 F 的查找线程仍在使用旧规则集。对于仍使用旧规则集的线程而言,只要在查找完 F 之前 F 的数据结构不被修改，则分类过程正确。

结点 F′ 替换 F 之后，F 即被回收。为避免查找线程离开 F 之前 F 的数据被修改，本文采用双链表显式管理内存的分配和回收。分配链表分配内存给更新线程使用，回收链表回收更新线程释放的内存。为防止更新线程立即使用刚回收的内存，回收链表中的内存不直接给更新线程使用，仅当经过一定次数更新之后才将回收链表中的内存并入分配链表。

4　实验结果及分析

4.1 实验设置

实验的硬件平台为带有 2 颗 Intel Xeon 5410/2.3GHz CPU、1333MHz 前端总线(FSB)和 4GB 内存的 Dell Power Edge 2900 服务器。每颗 Xeon 5410 有 4 个计算核，每核拥有 64KB L1 cache(指令和数据 cache 各 32KB)，每 2 个核共享 6MB L2 cache。操作系统为 GNU/Linux 2.6.26.5 SMP x86_64，编译器是 gcc 4.2.4，采用 Pthreads 来并行实现。

实验的规则集用 ClassBench[5]合成，ClassBench 提供了三种类型(ACL、FW 和 IPC)共 12 个种子(Seed)参数文件，本文选了五个种子文件来合成规则集。测试使用的包头文件用 ClassBench 工具生成，实验时从文件读入包头。

4.2 内存消耗与预处理时间

表 1　TIC 与 TICS 内存消耗和预处理时间对比

Size	内存消耗（MB）		预处理时间（S）	
	TIC	TICS	TIC	TICS
2K	3.51	0.21	34.01	0.03
4K	5.45	0.34	40.46	0.03

由表 1 知，TICS 的内存消耗比 TIC[4]降低了一个数量级，对于 4K 规则集只需要不超过 500KB。TICS 的预处理时间只有几十毫秒，比 TIC 降低了 3 个数量级。原因是递归空间分解中每条规则只保存到一个树结点中。

4.3 更新性能

表 2　TICS 的更新性能(Cycle)

Seed	2K 规则集		4K 规则集	
	Delete	Insert	Delete	Insert
ACL3	5040	5035	5847	5795
ACL4	3339	3337	5068	5071
ACL5	1296	1345	1564	1581
FW1	17032	16852	14251	14204
IPC1	3578	3591	5378	5216
平均	6057	6032	6422	6373

表 3　BRPS 的更新性能(Cycle)

规则集规模	1K	2K	3K	5K
Delete	56356	89257	16753	32950
Insert	69740	10718	18684	34774

表 2 是 TICS 只更新而无查找时的更新性能,表 3 是 BRPS 的更新性能[3]，数据均由一个线程获得，用平均每次更新操作所需时钟周期数来衡量。由表 2 和表 3 可知，TICS 更新性能较 BRPS 至少提升了一个数量级。对于 2K 规则集，BRPS 一次更新需约 10 万个周期，而 TICS 只需 6 千多个，减少了 15 倍。TICS 对于规则集规模的扩放性很好，从 2K 至 4K 规模，TICS 更新性能下降很少，而 BRPS 下降明显。

4.4 查找性能

表 4 是 TICS 只查找而无更新时不同查找线程数下每线程的平均分类速度，单位是百万兆数据包/秒(Mpps)，规则集规模为 2K。可看出，TICS 并行扩放性很好，几乎是线性加速。在并行结构上，良好的扩

放性可带来速度的明显提升。

表 4　TICS 无更新时不同线程数的分类速度(Mpps)

Seed	1T	2T	4T	6T	8T
ACL3	2.02	2.00	1.99	1.97	1.92
ACL4	1.78	1.77	1.75	1.72	1.70
ACL5	3.44	3.39	3.24	3.05	2.93
FW1	1.95	1.93	1.92	1.89	1.86
IPC1	2.09	2.08	2.07	2.01	1.99
平均	2.26	2.23	2.19	2.13	2.08

TICS 具有良好的并行扩放性的原因是其计算负载较大，并且数据结构所占内存很少，访存压力较小，这样可以充分利用多个计算核的计算资源来提升分类速度。

表 5 测试在查找和更新同时进行时对 TICS 算法性能的影响。规则集规模为 2K，查找速度单位为 Mpps，更新速度单位为时钟周期。全速更新是指不停顿地执行更新操作，表 5 中低于全速的更新是在更新操作之间停顿一定时间得到的，更新频率为次/秒。由表 5 可知，更新对查找性能有一定影响，更新频率越高，更新操作所需的时钟周期数越多，查找速度有所下降，但影响不是很大。全速更新与无更新相比，查找性能下降平均小于 20%。

表 5　不同更新频率下的查找和更新性能

更新频率	操作	1T	2T	4T	6T
347487 全速	Search	1.89	1.83	1.80	1.66
	Delete	8521	8951	9024	9933
	Insert	8852	9280	9413	10314
139775	Search	2.09	2.04	2.02	1.90
	Delete	8076	8232	8418	8739
	Insert	8329	8607	8994	9881
23023	Search	2.20	2.18	2.14	2.09
	Delete	7110	7374	7128	7850
	Insert	7252	7688	7769	8889
无更新	Search	2.26	2.23	2.19	2.13

5　结论

本文设计和实现了一个支持快速增量更新的两阶段五维包分类算法 TICS，其预处理时间短，消耗内存少，并行扩放性好，并实现了单写者-多读者的并行无锁增量更新。

参考文献

1 Buddhikot MM. Space Decomposition Techniques for Fast Layer-4 Switching. Proc. of IFIP. B.V. Deventer: Kluwer, 1999.25－42.

2 Taylor DE. Survey and taxonomy of packet classification techniques. ACM Computing Surveys, 2005, 37(3):238－275.

3 Chang YK. Efficient multidimensional packet classification with fast updates. IEEE Trans. on Computers, 2009, 58(4):463－479.

4 Cheng HP, Chen Z, Hua B, Tang XN. Scalable packet classification using interpreting-a cross-platform multi-core solution. Proc. of ACM SIGPLAN PPoPP. New York: ACM, 2008. 33－42.

5 Taylor DE, Turner JS. ClassBench: a packet classification benchmark. IEEE/ACM Trans. on Networking, 2005. 2068－2079.

6 Intel 64 and IA-32 Software Developer's Manual Vol. 3A: System Programming Guide, Part I, Intel, Nov. 2008.