基于交叉位图的多维流分类算法 *

胡茂福[1], 侯整风[1], 韩江洪[1], 何 玲[2]
(1. 合肥工业大学 计算机与信息学院, 合肥 230009; 2. 深圳金山信息安全技术有限公司, 广东 深圳 518057)

摘 要: 聚合位向量算法(ABV)是一种快速的流分类算法,但由于空间占用量大,并不适合大规模规则库。基于聚合位向量算法,提出一种新的改进算法,在不影响时间效率的基础上,通过改变算法中位图的存储方式,将聚合位图与位图交叉存储,忽略位图中全为 0 的部分,极大地减少了空间开销。最后,在仿真环境中对算法进行评测表明,该算法在大规模规则库中具有良好的时空效率。

关键词: 聚合位向量; 流分类; 位图; 交叉存储
中图分类号: TP393 **文献标志码**: A **文章编号**: 1001-3695(2010)08-3058-03
doi:10.3969/j.issn.1001-3695.2010.08.065

Algorithm of multi-dimensional flow classification based on intersecting bitmap

HU Mao-fu[1], HOU Zheng-feng[1], HAN Jiang-hong[1], HE Ling[2]
(1. *School of Computer & Information, Hefei University of Technology, Hefei* 230009, *China*; 2. *Shenzhen Kingsoft Information Security Technology LTD. Co, Shenzhen Guangdong* 518057, *China*)

Abstract: Aggregated bit vector is a fast scheme for flow classification, but it does not suit a large rule database because of the spending on space. By altering the way of storage, this paper presented a new algorithm. The new algorithm reduced the complexity of the storage by intersecting storage with aggregated bitmap and neglecting the part of zero, with maintaining the efficiency of time. In the end, realized the algorithm in an emulated environment and analyzed the scheme has good efficiency of time and space in a large rule database.

Key words: aggregated bit vector; flow classification; bitmap; intersecting storage

0 引言

随着网络流量迅速增长,流分类技术成为路由器、防火墙等许多网络设备的关键技术之一。自 20 世纪 90 年代中期流分类问题被提出以来,许多学者对流分类算法进行深入的研究,并取得了很大的进展。其中,1998 年 Lakshman 等人[1]提出了位并行的思想,由此产生了聚合位向量(ABV)[2]算法,该算法对多维数据包进行分类时,具有较好的时间效率,但其空间占有率较高。2003 年麻省理工学院(MIT)的 Li Ji 等人[3]提出位向量折叠(AFBV)思想,并应用到位并行算法中,减少了算法的空间占用,但算法中的冗余计算降低了算法的时间效率。因此,本文对聚合位向量(ABV)算法进行了深入的研究,提出了一种改进算法,该算法通过聚合位图与位图间隔存储和忽略位图中都为“0”的子串,减少算法的空间消耗,大大降低了算法的空间复杂度。

1 ABV 算法

1.1 ABV 算法的基本原理

ABV(aggregate bit vector)算法是基于 BV[1](bitvector)算法的一种改进快速分类算法。该算法适合于稀疏空间下的数据分类,特别适合于网络数据流的分类,被广泛应用于防火墙、路由器和 IDS 系统中。

网络数据流的分类属于多元分类方式,最典型的是通过源 IP 地址、目的 IP 地址、源端口、目的端口以及协议类型五元组[4]来构建数据分类的规则库。BV 算法分别针对每个元组进行匹配,根据各元组匹配的结果得到最终分类结果,从而将所分的数据划分到规则库中的某一类别中。BV 算法需要为该元组建立一个 Trie 树,Trie 树为二叉树,树的深度与元组取值的长度有关。Trie 树上的每个节点对应一个 BV 位图 M,其长度 L 与规则数相同,位图的第 i 位对应规则库的第 i 条规则,当该节点匹配第 i 条规则时,则位图的第 i 位取值为“1”,否则为“0”。因此,匹配规则库的过程是通过 Trie 树定位节点,然后匹配候选规则,即该节点位图中为“1”的对应规则。表 1 为规则二元组,由此得到如图 1 所示的 Field1 字段的 Trie 树,位图 M 的长度 $L=9$。

表 1 二元组规则库示例

Rules	Field1	Field2	Rules	Field1	Field2
R0	00 *	00 *	R5	0 *	00 *
R1	00 *	01 *	R6	1 *	00 *
R2	10 *	00 *	R7	11 *	10 *
R3	11 *	1 *	R8	10 *	1 *
R4	0 *	10 *			

收稿日期: 2009-12-26; **修回日期**: 2010-02-08 **基金项目**: 安徽省自然科学基金资助项目(090412051);广东省教育部产学研结合项目(2008B090500240)

作者简介:胡茂福(1984-),男,安徽来安人,硕士研究生,主要研究方向为计算机网络与信息安全(violet_mfh@163.com);侯整风(1958-),男,安徽和县人,教授,硕导,主要研究方向为计算机网络、信息安全;韩江洪(1954-),男,安徽泾县人,教授,博导,主要研究方向为智能控制、网络与信息系统;何玲(1962-),女,高级工程师,主要研究方向为信息安全.

由以上示例可看出,规则库中规则数越大,L 越大,匹配规则时间将越长。通过研究发现,在实际应用中一个节点匹配的规则很有限[5,6],因此 BV 位图 M 是稀疏的,即 M 中"1"的个数稀少。ABV 算法根据这一特点将稀疏 BV 位图 M 按聚合因子 a 进行聚集,即将 BV 位图 M 中 a 位聚合为 1 位,得到聚合位图 M′。如果 BV 位图 M 中的连续 a 位至少有一位为"1",则对应的 M′中聚合位为"1",否则为"0"。如图 1 所示,聚合因子 $a=3$,位图 M′的长度为 3,位图 M′长度小于 BV 位图 M。

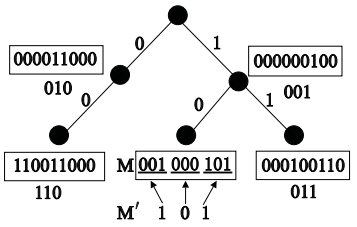

图1　Field1的Trie树

ABV 算法首先匹配 M′,当 M′中匹配到为"1"的位时,再进一步匹配 M 中对应的 a 位。通过位聚合的方式,ABV 算法减少了读取位图的位数,从而加快了算法执行,降低了匹配运算的复杂度。对于大规模规则库,采取两次聚合或多次聚合来压缩位串中的比特数目,可以减少查找的难度,提高效率。

1.2　ABV 算法的不足

ABV 算法通过位图聚合的方式,改善了 BV 算法在平均情况下的执行时间,但 ABV 算法的空间开销较大。在图 1 中,ABV 算法不仅要存储聚合位图 M′,还要同时存储位图 M,要比 BV 算法多占用 1/3 的空间。假设规则库规则数为 N,要存储位图的节点数目(节点数与规则库规模有关)为 C,聚合因子为 a,则 BV 算法的空间占用为 $C\times N$,而 ABV 算法为 $C\times N+C\times N/a$,因此,ABV 算法中对位图的存储要比 BV 算法多占用 $C\times N/a$。随着规则数目增加,C 也随之增加,ABV 算法的空间占用将激增。

2　ABV 算法的改进

2.1　位图交叉存储的思想

ABV 算法的存储方式是将 BV 位图和聚合位图分开存储,其目的是为了聚合位图和 BV 位图的对应,提高运算效率,但是将 BV 位图完全保存下来,需要大量的内存空间。在规则库中,能被节点匹配的规则数目有限,而且匹配成功的规则相对集中[7],所以 BV 位图段中有很多位是连续的"0",这些"0"占据了大量的空间,造成了空间的浪费。

针对全"0"位图段(位图段中每个比特位都为 0)占据大量空间的问题,本文提出了 BV 位图和聚合位图交叉存储的思想:首先,将 BV 位图按聚合因子聚合,产生聚合位图。然后,扫描聚合位图,若扫描位为"1",则在聚合位图后追加该聚合位对应的 BV 位图部分;若扫描位为"0",则跳过该位继续扫描下一位……最后形成交叉位图并保存(无须保存 BV 位图和聚合位图)。

设聚合因子为 8,有一长度为 $8\times n$(n 为整数)的 BV 位图向量:

$$(a_1,a_2,a_3,\cdots,a_n)$$

其中:a_i 的长度为 8,a_2、a_5、a_{10}不全为"0"。

则对应的聚合位图向量为

$$(b_1,b_2,b_3,\cdots,b_n)$$

交叉位图向量为

$$(b_1,b_2,b_3,\cdots,b_n,a_2,a_5,a_{10})$$

例如,设聚合因子为 8,有一个 64 bit 的 BV 位图 P 为

0000000001000100000000000000000000000000000000000000000000010000

则聚合位图 A 为

01000001

所以,BV 位图与聚合位图的总空间占用为 64 + 8 = 72 bit。

由位图交叉存储思想得出交叉位图 Q 为(实线部分为聚合位图):

010000010100010000010000

交叉位图空间占用为 24 bit,要比存储 BV 位图与聚合位图所需的空间占用量少 48 bit。

上例中,聚合因子为 8,扫描交叉位图 Q,Q 的聚合位图部分的第 2 位为"1",则查找位对应的 BV 位图部分(虚线部分),该部分的第 2 个比特位数值为"1",由此得出原始 BV 位图(形成交叉位图的 BV 位图)的第 $(2-1)\times 8+2=10$ 个比特位数值应为"1"。这与 BV 位图 P 的第 10 位数值相同,所以交叉位图能够还原 BV 位图,不影响位图查询能力。

2.2　改进的 ABV 算法

根据位图交叉存储的思想,本文提出了一种改进的 ABV 流分类算法。新算法分两部分,即预处理过程和查找过程。预处理过程对规则建树或表,为树节点和表元素指派一个位图,用来指示哪些规则对应该节点。在查找过程中,首先在预处理所建的各维树或表上并行搜索,查找出数据包各字段所对应的节点,由此得到在每一维上的位图;然后对所用位图进行与操作,得到数据包所对应的规则,对数据包按规则处理。

2.2.1　预处理过程

给定规则库 FD,假定它有 N 条 K 维规则,分别以 $F_1,\cdots,F_N$ 表示,$F_i(j)$ 表示规则 F_i 的第 j 维字段的值。改进的 ABV 算法的预处理过程如下:

a)对 FD 的每一规则 $F_i(1\leqslant i\leqslant N)$,将其源/目的 IP 字段分别建 Trie 树,对源/目的端口号和协议号投影到坐标轴上,与 ABV 算法相同,最多形成 $2N+1$ 个不相交的区间,将区间范围数值存储到顺序表。

b)为 Trie 树的有效节点(有位图对应的节点)和表元素指派一个 N 位的临时位图 B,B 中的每一位代表一条规则,B 的求取与 ABV 算法中位图求取相同。

c)为 Trie 树的有效节点和表元素映射一个交叉位图 IB。通过位图 B 按聚合因子 a 求出聚合位图 B′,扫描 B′,若扫描位为"1",则在聚合位图后追加该位聚合前对应的 BV 位图部分;若扫描位为"0",则跳过该位扫描下一位。最后,扫描 B′结束,形成交叉位图 IB,只保留 IB,释放 B、B′。

以表 1 所示的规则集为例,采用交叉存储方式存储每个节点对应的位图,聚合因子 a 为 3,得到如图 2、3 所示的 Field1、Field2 字段的 Trie 树,树中有效节点(有位图对应的节点)对应位图是交叉位图。交叉位图中下划线标注的比特位为聚合位图部分,在聚合位图后存储每个非零的聚合位对应的聚合前的位图。

2.2.2 数据包查找过程

设数据包为 $P(E_1,E_2,\cdots,E_k)$(E_i 为数据包中的字段,$1\leqslant i\leqslant k$),聚合因子为 a,那么规则的查询过程如下:

a)与 ABV 算法类似,首先在每一维 $j(1\leqslant j\leqslant k)$ 上通过对 Trie 树查找或二分查找求出 E_j 所对应的树中节点或表中位置,由此得出该数据包在第 j 维上对应的交叉位图 IB_j。

b)对于所有的交叉位图 $IB_1,\cdots,IB_k$ 的聚合位图依次按照机器读取长度 w 进行与操作。当第 i 次聚合位图与操作后没有出现比特位为"1"时,分别计算此次聚合位图段 $IB_{1i},\cdots,IB_{ki}$ 中比特位为"1"的个数 $C_1,\cdots,C_k$,并与上次的数值累加,然后读取相应长度的聚合位图继续进行与操作;当第 i 次聚合位图与操作后出现比特位为"1"时,分别求出该比特位的位置 p 及在聚合位图段 $IB_{1i},\cdots,IB_{ki}$ 中的位置 $S_1,\cdots,S_k$($IB_{ji}(1\leqslant j\leqslant k)$ 中可能有很多比特位为"1",S_j 记录该比特位在这么多"1"中所排的次序)。

c)分别取出交叉位图 $IB_1,\cdots,IB_k$ 中聚合位图部分后的第 $C_1+S_1,\cdots,C_k+S_k$ 处的位图段 $IB_{c1+s1},\cdots,IB_{ck+sk}$,将它们进行与操作,得到位图 B_t,B_t 中第一位为"1"的比特对应位置为 t,则该数据包对应规则数据库中第 $i\times w\times a+p\times a+t$ 条规则。

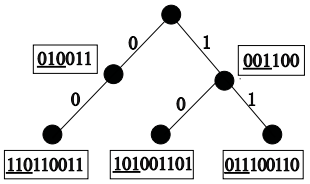

图2 Field1的Trie树

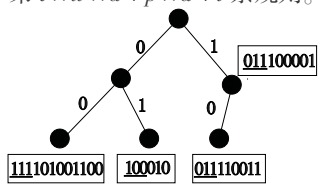

图3 Field2的Trie树

以数据包 $P(100,101)$ 为例。P 的两个字段分别在 Field1、Field2 的 Trie 树上查找。在 Field1 的树上查找到前缀为"10"的节点,它所对应的交叉位图 IB_0 为"101001101";在 Field2 的树上查找到前缀为"10"的节点,它所对应的交叉位图 IB_1 为"011110011"。将 IB_0、IB_1 按查找过程的步骤 b)进行与操作:设聚合位图与操作时每次读取的长度 w 为 3,则 IB_0、IB_1 的聚合位图只需读取一次,即 i 为 1。两个聚合位图"101"和"011"与操作后为"001""001"中第一位为"1"的比特对应位置 p 为 3,再按查找过程的步骤 c)进行规则查找。规则查找:由不为"1"的聚合位提取交叉位图中对应的 BV 位图部分"101"和"011",进行与操作,结果为"001""001"中第一位为"1"的比特对应位置 t 为 3;聚合因子 a 为 3,按 $i\times w\times a+p\times a+t$ 公式得到匹配规则序号为 $(1-1)\times3\times3+(3-1)\times3+(3-1)=8$(此处计数从 0 开始所以要减 1),即对应表 1 所示规则库中的 R8 规则。

2.2.3 改进算法的分析

设 N 是规则的数量,k 是规则维数,a 为聚合因子。

在空间性能上,改进算法与 ABV 算法的主要差别在于位图存储方式不同。设 C 为每维中要存储位图的节点数目,则 ABV 算法的总存储空间为 $k\times C\times(N+N/a)$ bit。改进算法采用交叉存储方式只存储位图中不全为"0"的段,在通常情况下不全为"0"的段较少,这样就达到了降低位图占用空间的目的。设 P 为改进算法中不为"0"的段的个数,则其存储空间为 $k\times C\times(P\times a+N/a)$ bit。最坏情况下 $P=N/a$,改进算法的空间占用与 ABV 算法相同。通常情况下,由于规则库中一个节点所匹配的规则数有限[5,6],$P<<(N/a)$,改进算法具有良好的空间性能。

在时间性能上,只考虑访问内存的次数,忽略每个字长中位的运算。设聚合位图相与后聚合位是"1"的个数为 M,即要读取的位图次数最多为 M。查找过程中,ABV 算法是先聚合位图与操作,当与操作结果中有"1"存在时,再读取相应位图,所以访问内存次数 $k\times(N/a+M)$。改进算法的操作方式与 ABV 算法相同,则其访存次数也为 $k\times(N/a+M)$ 次。因此,改进算法具有与 ABV 算法相同的时间效率。

3 算法评测

本文在仿真环境下对改进算法以及 ABV 算法进行评测。开发环境:Dell OPTIPLEX 210L(2.8 GHz Intel Pentium 4 处理器,1.24 GB 内存)PC 机、Ubuntu8.04 系统(系统内核为 Linux-2.6.24)。开发工具:gcc 编译器、gdb 调试器和 gedit 编辑器。在 Linux 下用标准 C 编程,实现改进算法,并通过在 netfilter[8] 框架的钩子处加载改进算法,构建仿真环境,用网络中的数据流来检测算法性能。

评测目的是对改进算法和 ABV 算法的分类速度与占用空间进行定量评测和对比。评测所用规则库的规则由程序生成,每条规则由源 IP 地址、目的 IP 地址、源端口、目的端口以及协议类型五元组组成。

3.1 评测结果

规则库规模分别是取 1k、2k、5k、10k、20k、30k,规则维数为 5,聚合因子为 8。由改进算法和 ABV 算法分别对规则进行预处理,由此得到位图存储占用的内存量。预处理结束后,对流经的数据包进行分类,统计出算法对数据包的处理速度。评测结果如图 4、5 所示。

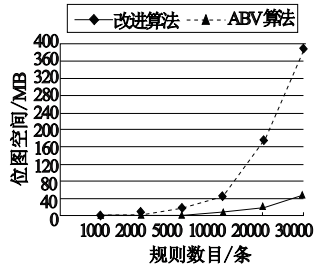

图4 算法的位图空间消耗

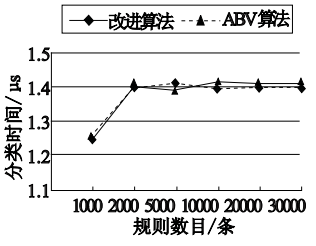

图5 算法的平均分类时间

3.2 评测数据分析

如图 4 所示,改进算法在空间占用上比 ABV 算法要少得多。实验中,随着规则库规模的增加,ABV 算法占用的内存量急剧增加。在 30 000 条规则的环境下,ABV 算法位图空间占用量已接近 400 MB,是改进算法位图空间占用量的近 10 倍。因此,改进算法在空间效率上较 ABV 算法有着明显的提升,特别是当规则库规模很大时,空间优势更加明显。

如图 5 所示,改进算法和 ABV 算法对数据包的处理速度基本相同,改进算法在时间效率上保持了 ABV 算法的高效性。

4 结束语

流分类算法通常在系统内核中运行,由于内核资源有限,对流分类算法空间性能的研究具有重要意义。ABV 算法在大规模规则库情况下,位图长度较大,空间复杂度增加。改进的 ABV 算法保持了 ABV 算法时间上的高效性,在不影响速度的前提下,采用交叉存储的方式存储原位图和聚合位图,大大降低了存储空间,避免了大规则库的空间激增,具有良好的时空效率,更适用于大规模规则库的分类。

(下转第 3063 页)

则,不真实。其中,τ 是区分恶意窜改与非恶意窜改的阈值。

3　实验结果

本文以 512×512 的标准 Lena 灰度图像进行实验,将加权后的脊波系数矩阵分成 127×127 的小块,则共有 64 块,最后生成的哈希长度 $N=64$。实验中取 $\tau=0.2$。

3.1　鲁棒性实验结果

根据图像哈希的鲁棒性要求,对于经过一些非恶意性的常规处理后,哈希值应该基本保持一致。如表 1 所示,本文算法对 JPEG 压缩、高斯噪声、高斯低通滤波、涂鸦等具有非常好的鲁棒性,同时还能抵抗小于 10°的旋转失真、小于 1/5 的剪切攻击。

表 1　鲁棒性实验结果

对图像的修改	δ	认证结果
100% JPEG 压缩	0.000 000	真实
加高斯噪声	0.029 297	真实
高斯低通滤波	0.021 875	真实
剪切小于 1/5	0.125 000	真实
剪切大于 1/5	0.203 125	不真实
旋转 10°	0.078 125	真实
旋转 20°	0.390 625	不真实
涂鸦	0.062 500	真实

3.2　敏感性实验结果

根据图像哈希的敏感性要求,当图像受到恶意窜改时,认证结果应该是不真实的。图 3 是用不同的图像恶意窜改 Lena 图像的实验结果,恶意窜改后所得到的哈希值截然不同(即 $\delta>0.2$),认证结果显示为不真实,因此该算法对恶意窜改具有较好的敏感性。

(a) δ=0.265 625　　(b)δ=0.234 375

图3　恶意窜改实验结果

3.3　安全性实验分析

图像哈希方法的安全性要求给定一个图像的哈希算法,攻击者在没有密钥的情况下不能够预测图像的哈希值。如表 2 所示,采用不同的密钥得到的哈希值截然不同。所以本文算法具有较好的安全性。

表 2　采用错误密钥实验结果

错误密钥序号	δ	认证结果
1	0.515 625	无法预测
2	0.453 125	无法预测
3	0.359 375	无法预测
4	0.390 625	无法预测
5	0.484 375	无法预测

4　结束语

本文提出了一种新的基于视觉特性的图像哈希方法。利用人眼视觉特性,提取脊波域各子带系数作为特征,在密钥的控制下进行合理的量化、压缩。生成的哈希对 JPEG 压缩、滤波、噪声、涂鸦、10°以下的旋转、20% 以下的剪切具有很好的鲁棒性,并能区分恶意窜改。同时在密钥控制下生成哈希值,算法获得了较高的安全性。

参考文献:

[1] TIRKEL A Z, RANKIN G A, SCHYNDEL R M, *et al*. Electronic watermark digital image computing[C]// Proc of Technology and Applications(DICTA'93). Sidney:[s. n.],1993:666-673.

[2] FRIDRICH J, GOLJAN M. Robust hash functions for digital watermarking[C]// Proc of International Conference on Information. Technology: Coding and Computing. 2000:173-178.

[3] VENKATESAN R, KOON S M, JAKUBOWSKI M H, *et al*. Robust image hashing[C]// Proc of IEEE International Conference on Image Procssing. 2000: 664-666.

[4] MIHCAK M K, VENKATESAN R. New iterative geometric metheods for robust perceptual image hashing[C]// Proc of ACM Workshop Security and Privacy in Digital Rights Management. 2001:13-21.

[5] SWAMINATHAN A, MAO Yi-nian, WU Min. Rubust and secure image hasing[J]. IEEE Trans on Information Forensics and Security, 2006, 1(2):215-230.

[6] MONGA V, MIHCAK M K. Robust image hashing via non-negative matrix factorizations[C]// Proc of IEEE International Conference on Acoustics. 2006:3453-3466.

[7] 张金沙. 基于有限脊波变换的图像水印算法研究[D]. 成都:电子科技大学,2006.

[8] 肖亮,韦志辉,吴慧中. 基于图像内容的脊波变换域数字水印模型和算法研究[J]. 电子与信息学报,2004, 26(9):1440-1448.

[9] 邹建成,周红丽,邓欢军. 一种安全鲁棒的图像哈希方法[J]. 计算机应用研究, 2009, 26(6):2122-2124.

(上接第 3060 页)

参考文献:

[1] LAKSHMAN T V, STIDIALIS D. High-speed policy-based packet forwarding using efficient multi-dimensional rang matching[C]//Proc of ACM SIGCOMM. New York: ACM Press,1998:191-202.

[2] BABOESCU F, VARGHESE G. Scalable packet classification[J]. IEEE/ACM Trans on Networking,2005,13(1):2-14.

[3] LI J, LIU H Y, SOLLINS K. Scalable packet classification using bit vector aggregating and folding[D]. Cambridge: MIT Laboratory for Computer Science, 2003.

[4] WOO T. A modular approach to packet classification: algorithms and results[C]//Proc of IEEE INFOCOM. 2000: 1213-1222.

[5] TAYLOR D E. Survey and taxonomy of packet classification techniques[J]. ACM Computing Surveys,2005,37(5):238-275.

[6] TAYLOR D E, TURNER J S. ClassBench: a packet classification benchmark, WUCSE-2004-28[R]. Saint Louis: Department of Computer Science & Engineering, Washington University, 2004.

[7] TAYLOR D E, TURNER J S. Scalable packet classification using distributed crossproducting of field labels, WUCSE-2004-38[R]. Saint Louis: Department of Computer Science & Engineering, Washington University, 2004.

[8] RUSTY R. Linux 2.4 Packet Filtering HOWTO[EB/OL]. [2002-02-19]. mailing list netfilter@lists.samba.org.