一种面向连接的快速多维包分类算法

张 斌 吴浩明*

(中国人民解放军战略支援部队信息工程大学 郑州 450001)

(河南省信息安全重点实验室 郑州 450001)

摘 要：为进一步提高聚合位向量(ABV)算法分类数据包的速度，该文提出一种面向连接的改进ABV(IABV)算法。该算法利用同一连接包分类查找规则相对一致的特点，建立哈希表-规则库两级优化查找结构，首先通过哈希表查找包分类规则，若未命中继续从规则库中查找。利用连接时效性特点设计哈希表冲突处理机制，根据表项最近命中时间判断是否进行覆写更新，避免规则累积导致查找时间增加；其次对ABV算法各维度进行等分处理，为各等分区间建立数组索引，从而快速缩小向量查找范围，加快查找规则库速度；最后，将规则中前缀转化为范围降低辅助查找结构复杂度，以减少内存空间占用量并加快规则查找速度。实验结果表明，将规则中前缀转化为范围后能够有效提升算法性能，相同条件下IABV算法相比ABV算法时间性能有显著提高。

关键词：包分类；聚合位向量算法；哈希表；维度切分

中图分类号：TN919; TP391 **文献标识码**：A **文章编号**：1009-5896(2020)06-1526-08

DOI: 10.11999/JEIT190434

A Connection-oriented Fast Multi-dimensional Packet Classification Algorithm

ZHANG Bin WU Haoming

(*PLA Strategic Support Force Information Engineering University*, *Zhengzhou* 450001, *China*)

(*Henan Key Laboratory of Information Security*, *Zhengzhou* 450001, *China*)

Abstract: In order to increase the classification speed of Aggregated Bit Vector (ABV) algorithm, an Improved Aggregated Bit Vector (IABV) algorithm is proposed, which is connection-oriented. Based on the characteristic that the packets which belong to the same connection have similar classification results, IABV establishes a Hash table-rule set two-level searching structure. It first searches in the Hash table to check the packet classification rule and then finds the matching rule in the rule set when the Hash table lookup fails. To avoid the accumulation of rules in the table, a collision handling mechanism is proposed. It judges whether to overwrite the Hash table entry which is collision according to the last hit time of the entry; Secondly, for the purpose of accelerate rule set searching, IABV divides each dimension into multiple intervals equally and employs array to index these intervals; Finally, the prefix in the rule is converted into range to reduce the complexity of the search structure, so that the time and memory consumption of the algorithm can be decreased. The experiment result shows that the performance of the algorithm can be improved by converting prefix into range and the time performance of IABV algorithm is significantly improved compared with the ABV algorithm under the same conditions.

Key words: Packet classification; Aggregated Bit Vector (ABV) algorithm; Hash table; Dimension cutting

收稿日期：2019-06-13；改回日期：2020-03-03；网络出版：2020-03-27

*通信作者： 吴浩明 wuhaoming0512@126.com

基金项目：河南省基础与前沿技术研究计划基金(142300413201)，信息工程大学新兴科研方向培育基金(2016604703)，信息工程大学科研项目(2019f3303)

Foundation Items: The Foundation and Frontier Technology Research Project of Henan Province (142300413201), The New Research Direction Cultivation Fund of Information Engineering University (2016604703), The Research Project of Information Engineering University (2019f3303)

1　引言

包分类是指根据给定的规则库，将数据包分类为特定的流的过程[1]，已在路由器、防火墙、入侵检测等领域得到了广泛的应用[2,3]。近年来，随着数据中心网络、软件定义网络等前沿网络技术的发展，对数据包分类算法性能提出了更高的要求[4]。

位向量(Bit Vector, BV)算法[5]是经典的包分类算法，该算法将包分类过程划分为各个字段独立的查找过程，具有良好的并行能力和维度扩展能力，字段的增加不会显著增加其时间效率[6]。

目前对BV算法的研究大致可以分为2类：第1类将BV算法与各硬件平台进行结合，对BV算法的并行处理实现方式进行探讨。如基于多核平台对BV算法不同实现方式进行对比分析[7]；基于GPU对算法并行运算进行研究[8]等。这类方法侧重于依托硬件平台对算法的具体实现进行分析与设计，算法实现速度较快，但硬件实现成本较高，对算法自身改进考虑不多。第2类侧重于改进BV算法的设计机理。如流量自适应的多维包分类算法[6]、聚合位向量(Aggregated Bit Vector, ABV)算法[9]、改进的位向量流分类算法[10]等。该类算法对BV算法进行了深入分析，在一定程度上提高了算法的时间、空间性能，但算法没有充分利用同一连接包分类规则相对一致的特点，处理时间与规则数量紧密相关，算法执行效果较为依赖规则库结构。

为了进一步提高算法分类数据包的速度，本文对ABV[6]算法向量查找以及规则合成两个环节进行深入分析，提出了一种改进ABV(Improved Aggregated Bit Vector, IABV)算法。首先基于同一连接包分类查找规则相对一致的特性，建立哈希表-规则库两级优化查找结构，加快查找规则的速度；然后建立哈希表冲突处理机制，对ABV算法各维度进行等分处理，并在规则前缀域建立范围树以减少算法时间、空间消耗；最后通过对比实验验证所提算法的有效性。

2　ABV算法

规则库包含N条规则R_1,R_2,$\cdots$,R_N，已经按照优先级降序进行排列，每条规则包括d个维度，规则$R_i(i=1,2,\cdots,N)$表示为$(r_{i1},r_{i2},\cdots,r_{id})$，其中$r_{ik}(k=1,2,\cdots,d)$是一个取值范围。ABV算法处理过程如下。

2.1　ABV算法预处理

(1) 建立辅助查找数据结构：将所有规则第k维$r_{ik}(i=1,2,\cdots,N)$投影到数轴上，最多形成$2N+1$个不重叠区间$\mathrm{Iv}_{km}(m=1,2,\cdots,2N+1)$，建立合适的数据结构索引$\mathrm{Iv}_{km}$。

(2) 建立位向量：对每个Iv_{km}，建立N比特位向量$\boldsymbol{BV}_{km}$，若$\mathrm{Iv}_{km}\subseteq r_{ik}(i=1,2,\cdots,N)$，则$\boldsymbol{BV}_{km}$中第$i$个比特为1，否则为0。

(3) 建立聚合位向量：设置聚合度为L，将$\boldsymbol{BV}_{km}$中每L位聚合为1位，构建聚合位向量$\boldsymbol{ABV}_{km}$。若L位全为0，则聚合为0，否则为1。

2.2　ABV算法查找过程

(1) 向量查找：通过各维建立的数据结构找到包头d个字段分别所属的Iv_{km}，取出$\boldsymbol{BV}_{km}$与$\boldsymbol{ABV}_{km}$。

(2) 规则合成：将d个$\boldsymbol{ABV}_{km}$按位与，结果记为P，P中第1个比特1为P中第$x(x=1,2,\cdots,\lceil N/L\rceil)$位。取出$d$个$\boldsymbol{BV}_{km}$中第$L\cdot(x-1)+1$位到$L\cdot x$位按位与，结果记为$Q_x$，若$Q_x$不为0，$Q_x$中第1个比特1为$Q_x$中第$c(c=1,2,\cdots,L)$位，则将规则$R_{L(x-1)+c}$作为查找结果返回；否则继续寻找$P$中下一个比特1进行操作。

2.3　ABV算法分析

ABV算法将包分类问题分解为各维度对位向量和聚合位向量的查找问题，通过这两个向量计算出数据包匹配规则。这一过程中ABV算法以单个数据包为分类对象，主要利用规则的分布特点提高分类速度，但对数据包之间的关系考虑较少。其次ABV算法要求在各维度建立合适的辅助查找数据结构，对位向量与聚合位向量的查找时间与建立的数据结构有关。由此分析可知，若能对数据包之间的关系加以利用，根据前期数据包分类结果减少后续数据包分类时间，并对各维度使用的辅助查找数据结构进行优化，可有效提高包分类效率。

3　IABV算法

3.1　算法改进思想

3.1.1　自适应哈希表

属于同一个连接的数据包，报文分类结果相对一致[11]。利用这一点，使用算法从规则库中找到规则后，根据连接信息将规则写入到哈希表中，之后接收到数据包后首先访问哈希表查找规则，若查找成功则直接返回规则；否则再执行算法从规则库中查找规则，然后返回找到的规则并将该规则添加到哈希表中。

但哈希表更新时存在哈希冲突问题，使用通用的哈希冲突处理方法会导致规则在哈希表中不断累积，而且连接具有时效性，这些累积的规则在对应的连接结束后失去效用，严重降低查找效率。因此，不能不加区分地将所有连接匹配的规则都保存在哈希表中。

对于上述问题，本文提出根据哈希表表项最近

命中时间与当前时间之差度量表项的有效性，设置查找命中时间间隔阈值，该时间差小于阈值说明对应表项最近被使用过，之后继续命中的概率大，应该保留在哈希表中，哈希冲突时放弃写入操作；而时间差高于阈值则说明该表项对应的连接很可能已经结束，或者该连接当前发包速率低，表项作用有限，在哈希冲突时直接覆写。

上述方法中哈希表的大小以及阈值的设置十分关键。哈希表中保存的规则数量与哈希表大小有关，哈希表越小表中保存的规则数量就越少，查找哈希表失败的机率就越大，因此哈希表大小需要在实际情况允许范围内设置尽可能大的数值。为结合实际情况对阈值进行设置，本文对收集自商业网络骨干链路的CAIDA trace 2016(Center for Applied Internet Data Analysis)数据集[12]子集进行统计分析。由于数据集中56.9%的流持续时间小于10^{-6} s，数据包数目小于2，因此对持续时间不足1 s的流以1 s计算，数据集中流速(包每秒，pktps)前4.2%的流包含的数据包占据了数据包总数的69.2%，流速大于10 pktps。也就是说这些流包含的数据包之间平均时间间隔小于0.1 s，因此，文中将查找命中时间间隔阈值设置为0.1 s。

3.1.2 维度切分

ABV算法在各维度建立不同的数据结构辅助查找位向量与聚合位向量，通常对使用前缀匹配方式的维度建立特里树，使用范围匹配方式的维度建立范围树，使用精确值匹配的维度建立哈希链表。这3种数据结构的查找速度与其复杂度有关，如树结构查找时间取决于树的深度，如果能够降低树的深度，就能有效减少查找时间。

将整个维度e等分(e为小于等于维度长度的正整数)，为各等分区间建立数组进行索引，在投影区间分布较为均匀的情况下，每个等分区间内包含的投影区间数将少于投影区间的总数，在等分区间上建立的数据结构复杂度也将降低。e值越大显然每个等分区间内包含的投影区间数就越少，建立的数据结构也就越简单，当e等于维度长度时，每个等分区间是数轴上的一个点，此时每个等分区间内只包括一个投影区间，这种情况下对数组的一次访问就能查找到相应的位向量。

以图1(a)所示特里树为例，对整个维度4等分(前缀形式的维度进行2^f等分，f为小于或等于维度位宽的正整数，避免切分后区间难以表示为前缀的问题)，等分区间以前缀表示分别为00*, 01*, 10*和11*, *号为通配符，表示0或1，在等分区间内建立子特里树，结构如图1(b)所示。

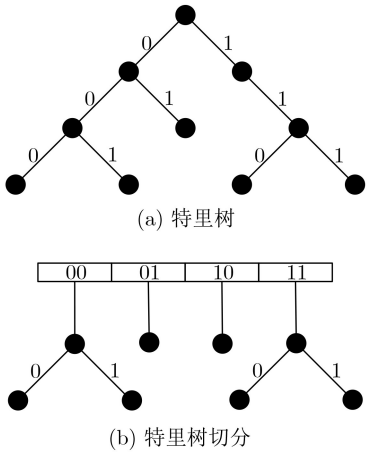


图 1 特里树切分示例

图1(a)中特里树深度为4，最多需要查找4次才能找到叶节点；而图1(b)中的结构最大深度为3，最多查找3次就能找到叶节点。

假设需要在图1(b)中查找111，首先根据前两比特找到数组11单元，然后通过数组中存储的指针找到相应的子树，最后根据第3比特找到子树中对应的1分支节点完成查找。

上述特里树切分中需要注意两个问题：(1)数组每个单元表示的前缀长度固定，例如图1(b)每个单元表示的前缀长度为2，而原特里树中存在路径长度小于2的节点，切分之后就丢失了这些节点，如路径为0的节点。这对最长前缀匹配方式没有影响，因为查找时不会停留在路径为0的节点，而会继续查找00或01节点。(2)数组所表示的前缀可能在前缀树中不存在。例如图1(b)中数组10单元，图1(a)中不存在路径为10的节点，对这种情况取10节点在特里树中最长前缀匹配节点(这里取路径为1的节点)建立10节点，因为匹配10节点的数据包也一定匹配10节点的前缀节点。

3.2 IABV算法流程

根据上述改进思想，建立IABV算法流程如图2所示。

(1) 预处理：将规则各维投影到数轴上，并对数轴进行e等分处理，在各等分区间上根据规则的投影区间建立辅助查找数据结构进行索引，建立数组索引各等分区间数据结构，根据规则库对位向量、聚合位向量进行设置。建立大小为S的哈希表，对哈希表每个单元设置最近命中时间$t_j(j=1,2,\cdots,S)$，设置命中时间差阈值σ以及时钟clk，最后开启时钟。

(2) 哈希表查找：接收到数据包后，取出数据包包头字段计算哈希地址j，访问哈希表读取规则。若哈希表内存储的规则与数据包匹配，则返回该规则并将clk赋值给t_j，否则进行后续规则库查找过程。

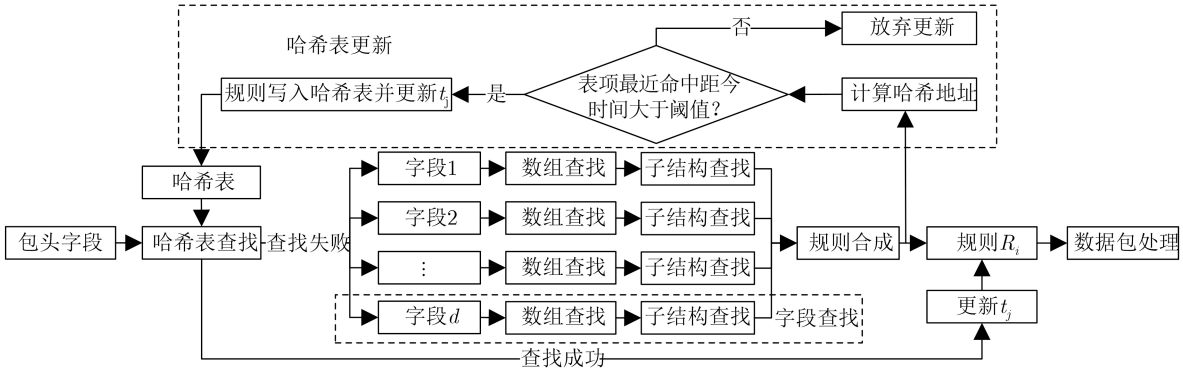


图 2 IABV算法流程

(3) 字段查找：取出数据包包头字段，各字段独立查找(串行或并行)。首先根据字段值访问数组，计算字段值所在的数组单元，找到对应辅助查找数据结构；然后访问该查找结构找到位向量和聚合位向量。

(4) 规则合成：根据位向量和聚合位向量找到与数据包匹配的规则并返回；若没有规则与数据包匹配，则返回默认规则。

(5) 哈希表更新：从规则库中查找到规则后，根据包头字段哈希值找到哈希表表项，计算clk $-\ t_j$，判断是否进行写入。若clk $-\ t_j \geq \sigma$，则写入规则，clk赋值给t_j；否则，放弃此次操作。

(6)数据包处理：根据算法返回的规则对数据包进行处理。

IABV算法伪代码如表1所示。

4　算法及实验分析

4.1　算法分析

4.1.1　时间复杂度

IABV算法最差情况下相当于在ABV算法查找规则库基础上增加2次对哈希表的访问，由于访问哈希表时间为常量，因此IABV算法最差情况下时间复杂度与ABV算法相当。设内存访问位宽为W，算法聚合度为L，则最差情况下IABV算法时间复杂度为$O\left(N/W+N/\left(LW\right)\right)$。IABV算法最好情况下访问哈希表直接找到匹配规则，其时间复杂度为$O(1)$。

4.1.2　空间复杂度

ABV算法的空间复杂度为$O(dN^2)$[13], IABV算法在ABV算法的基础上建立了哈希表并对维度进行了切分，哈希表大小为S，各维度进行e等分，则IABV算法空间复杂度为$O(dN^2+S+de)$。

4.2　实验分析

实验环境为64位Windows7操作系统的台式计算机，内存为8 GB，处理器为Intel(R) Core(TM) i5-4590，程序使用C++语言编写，规则库和数据集由ClassBench[14]产生。ClassBench是包分类领域广泛使用的规则产生器，可以生成ACL(Access Control List), FW(Fire Wall)和IPC(Linux IP Chains) 3种类型的规则。

实验使用的规则库IP域为前缀形式，端口域为范围形式，协议域为精确值。实验中对IP域分别建立了步长为1, 2和4的特里树(分别以Trie-1, Trie-2和Trie-4表示)以及将前缀转化为范围后建立范围树(以RTree表示)，对比分析不同结构的性能；对端口域采取建立范围树的方式，由于将范围转化为前缀最差情况下会导致规则数扩张30倍[15]，因此没有将范围转化为前缀；协议域使用了哈希表查找，采取了链地址法处理哈希冲突。BV, ABV和IABV算法各维度的查找采用串行方式实现。

ABV和IABV算法聚合度设置为32。IABV算法哈希表大小设置为1000, σ设置为0.1，对IP域以及端口域进行了256等分(协议域包含的值较少，不进行切分)。

4.2.1　预处理时间

表2是IABV算法，ABV算法和BV算法在不同规则库下，IP域使用不同辅助查找数据结构的预处理时间。IABV算法由于额外进行了建立哈希表与维度切分操作，需要花费更多时间进行预处理。在相同规则库下，IABV算法相比ABV算法预处理时间增加0.3%～8.6%，相比BV算法增加0.5%～11.0%。

对比IABV算法IP域分别使用4种数据结构的预处理时间，Trie-2相比Trie-1, ACL规则10 k规模下预处理时间增加7.0%, 20 k和30 k分别减少6.5%, 0.1%, FW规则和IPC规则预处理时间减少37.6%～41.7%; Trie-4相比Trie-1, ACL规则预处理时间分别增加15.4%, 1.2%, 22.0%, FW规则和IPC规则预处理时间减少39.0%～45.9%；使用RTree时的预处理时间最短，相比Trie-1, ACL规则预处理时间分别减少8.6%, 20.0%, 3.2%，而FW规则与IPC规则的预处理时间减少70.0%～77.2%。通过增加特里树

表 1 IABV算法伪代码

输入：数据包Packet, 规则库$R_{1,2,\cdots,N}$

输出：数据包匹配的规则Rule

(1) Axis$[1,2,\cdots,d]$= Project$(R_{1,2,\cdots,N})$//将所有规则d个维度分别投影到数轴上;

(2) AxisCut$[1,2,\cdots,d][1,2,\cdots,e]$= CutDimension(Axis$[1,2,\cdots,d],e$)//将各维$e$等分;

(3) FOR $m=1$ TO d;

(4) FOR $n=1$ TO e;

(5) Array$[m][n]$ = BuildSearchStructure(AxisCut$[m][n]$)//对各维等分区间建立相应数据结构对其内的投影区间进行索引，并建立数组索引各维等分区间上的数据结构;

(6) BuildVector(Array$[m][n]$, $R_{1,2,\cdots,N}$)//根据规则库中规则对位向量以及聚合位向量进行设置;

(7) END FOR

(8) END FOR

(9) Build(HashTable$[1,2,\cdots,S],t[1,2,\cdots,S],\sigma$, clk)//建立哈希表与时钟，对哈希表每个单元设置最近命中时间$t_j(j=1,2,\cdots,S)$，设置命中时间差阈值σ并开启时钟clk;

(10) WHILE PacketArrive()//接收到数据包时进行操作;

(11) Field$[1,2,\cdots,d]$ = GetField (Packet)//取出d个包头字段;

(12) j = Hash (Field$[1,2,\cdots,d]$) % $(S+1)$//计算哈希地址;

(13) IF Field$[1,2,\cdots,d]\in$ HashTable$[j]$//若哈希表中存储的规则与数据包匹配，更新命中时间并返回规则;

(14) t_j= clk;

(15) RETURN HashTable$[j]$;

(16) END IF

(17) FOR $i=1$ TO d//d个维度分别查找向量;

(18) Root$[i]$ = Array$[i]$ $[\lceil e*$ Field$[i]$/Range(Field$[i]$)$\rceil]$//计算字段值所在的数组单元，取出子结构根结点;

(19) ABV$[i][1,2,\cdots,\lceil N/L\rceil]$, BV$[i][1,2,\cdots,N]$ = FindVector(Root$[i]$, Field$[i]$)//查找子结构，找到位向量与聚合位向量;

(20) END FOR

(21) P = ABV[1]&$\cdots$&ABV$[d]$//聚合位向量按位与;

(22) FOR $x=1$ TO P.Length//对P中每一位执行循环;

(23) IF $P[x]=1$//如果P中第x位为比特1;

(24) Q_x = BV[1]$[L\cdot(x-1)+1, L\cdot(x-1)+2,\cdots,L\cdot x]$&$\cdots$&BV$[d][L\cdot(x-1)+1, L\cdot(x-1)+2, \cdots, L\cdot x]$//取出BV中与$x$对应的$L$位按位与;

(25) IF $Q_x!=0$;

(26) $c=Q_x$.FirstBitPosition(1)//Q_x中第1个比特1的位置;

(27) IF clk$-t_j\geq\sigma$//判断规则是否写入哈希表;

(28) HashTable$[j]=R_{L(x-1)+c}$;

(29) t_j = clk;

(30) END IF

(31) RETURN $R_{L(x-1)+c}$;

(32) END IF

(33) END IF

(34) END FOR

(35) RETURN DefaultRule//返回默认规则;

(36) END WHILE

步长或建立范围树方式可有效减少算法在FW规则和IPC规则下预处理时间，而ACL规则由于IP域前缀较“精确”，长度普遍大于16，需要建立的位向量、聚合位向量较少，因此其预处理时间变化不大。

综上所述，IABV算法预处理时间较ABV算法略有提升，但算法执行过程中只需要进行一次预处理，对算法总体性能影响不大；对于这4种数据结构，IP域使用RTree时的预处理时间最短。

表 2　算法预处理时间对比(ms)

数据结构		规则库规模								
		10 k			20 k			30 k		
		ACL	FW	IPC	ACL	FW	IPC	ACL	FW	IPC
Trie-1	BV	1174	12540	42496	3699	106915	195698	5075	202435	430327
	ABV	1192	12602	45231	3798	110567	196068	5125	209298	440421
	IABV	1237	12659	45563	4101	111206	196624	5163	209871	443620
Trie-2	BV	1201	7083	25146	3453	66191	118100	4987	126026	266448
	ABV	1262	7390	26131	3531	67828	118219	5043	129580	270810
	IABV	1323	7536	26579	3834	68622	118793	5158	131054	272014
Trie-4	BV	1368	7182	23571	3973	62380	105742	6124	120220	242206
	ABV	1396	7503	24300	4055	64571	106346	6233	123834	250471
	IABV	1427	7620	24667	4151	66576	111140	6301	127998	255087
RTree	BV	1069	3448	10514	3207	28270	43799	4905	56598	106005
	ABV	1117	3589	10675	3273	28775	44428	4910	57610	106186
	IABV	1131	3793	11095	3284	29200	44890	4996	59252	107847

4.2.2　内存访问次数

由于访问内存相比访问寄存器需要消耗更多的时间，包分类算法一般以内存访问次数来衡量算法的查找速度。分别在10 k, 20 k和30 k规模的ACL, FW和IPC规则库下对使用4种数据结构的BV, ABV和IABV算法进行了测试，记录了最小内存访问次数，最大内存访问次数和平均内存访问次数如表3所示。

最小内存访问次数体现了自适应哈希表快速查找规则的作用。IABV算法最快情况下直接在哈希表找到数据包匹配规则，最小内存访问次数都为5，相比ABV算法和BV算法减少81.5%～94.0%。

最大内存访问次数是实验中出现的最差情况，一定程度上体现了算法最差情况下的性能。IABV算法最差情况下相比ABV算法增加了哈希表读写操作，表3中IABV算法最大内存访问次数相较ABV算法平均增加0.5%。但使用Trie-1和RTree时，IABV算法在大部分规则库下相比ABV算法的最大内存访问次数都有所减少，这是由于维度切分机制减少了查找向量需要访问内存的次数，使用Trie-1与RTree时该机制效果更好。

平均内存访问次数主要反映算法的平均性能。IABV算法平均内存访问次数相比ABV算法减少42.3%～79.7%，平均减少56.8%，相比BV算法减少60.7%～96.0%，平均减少89.1%。对比使用4种数据结构时IABV算法的平均内存访问次数，使用Trie-4时平均内存访问次数最小，相比Trie-1平均减少11.1%；相比Trie-2平均减少4.4%；相比RTree，在ACL规则库下平均内存访问次数相同，FW和IPC规则库下平均减少2.4%。

IABV算法平均性能有较大幅度的提升，这是由于采用自适应哈希表使大量数据包不必执行复杂的规则库查找过程，在访问内存5次的情况下就能完成包分类过程，同时在查找哈希表失败需要查找规则库的情况下，通过维度切分减少了对规则库的查找时间，这两点改进使得IABV算法在平均内存访问次数上优于ABV和BV算法。其次在前缀域通过增加特里树步长或建立范围树两种方式降低了树的最大深度和平均深度，从而减少对树结构的查找次数，提升了算法时间性能。

综上，使用相同辅助查找数据结构与规则库时IABV算法时间性能优于ABV算法和BV算法，4种数据结构中Trie-4和RTree时间性能较好。

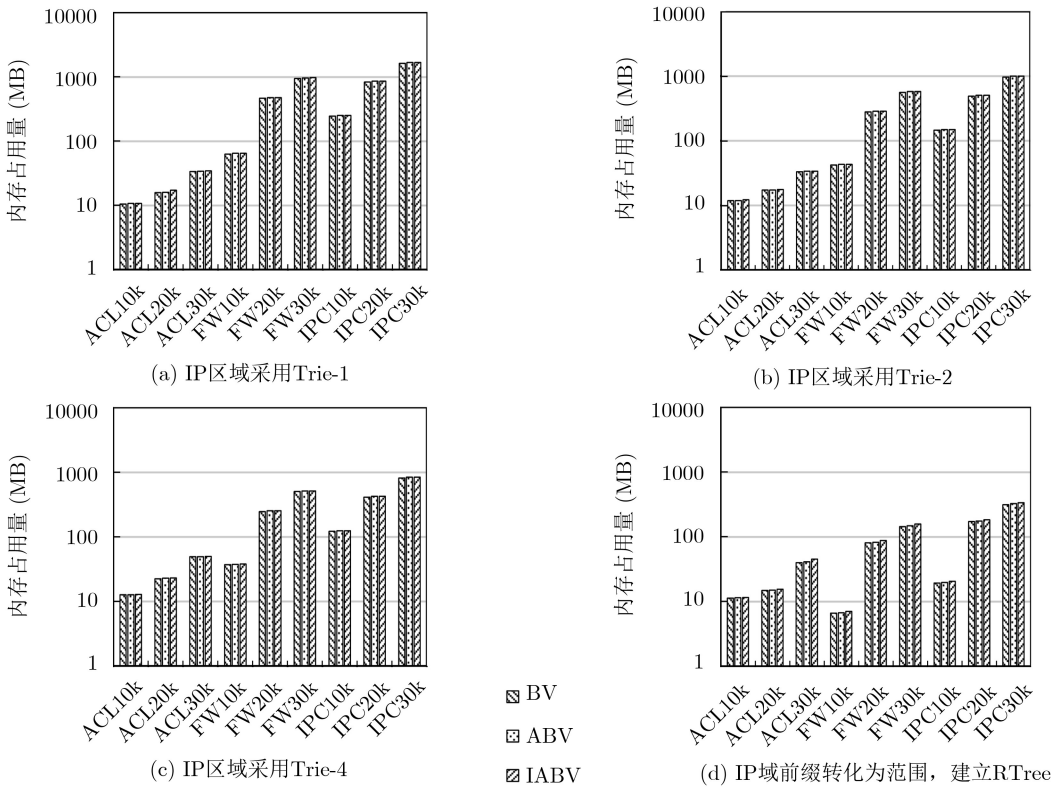
4.2.3　内存占用量

图3显示了使用4种数据结构的IABV, ABV和BV算法在不同规则库下分别需要占用的内存量。实验结果表明IABV算法相比ABV算法，内存占用量增加0.1%～7.5%；相比BV算法，内存占用量增加1.7%～8.7%, IABV算法的空间性能有所下降。

由图3可见，随着特里树步长的增加，算法使用ACL规则时内存占用量有所增加，使用FW规则和IPC规则库时占用的内存量减少。以IABV算法为例，Trie-2相比Trie-1, ACL 10k和20k规则内存占用量分别增加13.0%和2.1%, ACL 30k规则内存占用量减少1.7%, FW规则和IPC规则内存占用量减少33.4%～41.2%。Trie-4相比Trie-1, ACL规则的内存占用量分别增加18.8%, 35.4%和45.1%, FW规则和IPC规则内存占用量减少41.6%～50.6%。而RTree相比Trie-1，除ACL 10k规则内存占用量增加7.6%, ACL 20k和30k规则分别减少9.9%和

表 3 算法内存访问次数对比

规则库	Trie-1									Trie-2								
	最小内存访问次数			最大内存访问次数			平均内存访问次数			最小内存访问次数			最大内存访问次数			平均内存访问次数		
	BV	ABV	IABV	BV	ABV	IABV	BV	ABV	IABV	BV	ABV	IABV	BV	ABV	IABV	BV	ABV	IABV
ACL 10k	78	78	5	1103	471	466	605	110	48	46	51	5	1071	450	453	579	83	40
ACL 20k	78	83	5	3273	923	919	1524	137	70	46	51	5	3241	891	895	1498	111	61
ACL 30k	46	48	5	4638	1018	1014	2181	177	88	31	35	5	4606	986	990	2154	151	79
FW 10k	41	30	5	292	278	280	145	121	57	30	27	5	267	261	268	130	106	53
FW 20k	75	57	5	3271	2256	2251	1608	621	240	46	39	5	3260	2248	2248	1592	605	236
FW 30k	75	56	5	4720	2858	2853	3045	965	197	47	39	5	4715	2852	2852	3031	951	194
IPC 10k	43	48	5	1614	658	664	824	244	75	29	34	5	1596	644	654	803	223	71
IPC 20k	73	59	5	3163	733	735	1614	219	89	43	45	5	3149	719	727	1592	197	84
IPC 30k	70	64	5	4755	1092	1097	2280	386	203	42	46	5	4727	1078	1088	2258	365	196
规则库	Trie-4									RTree								
	最小内存访问次数			最大内存访问次数			平均内存访问次数			最小内存访问次数			最大内存访问次数			平均内存访问次数		
	BV	ABV	IABV	BV	ABV	IABV	BV	ABV	IABV	BV	ABV	IABV	BV	ABV	IABV	BV	ABV	IABV
ACL 10k	30	35	5	1055	439	446	566	70	36	29	34	5	1054	441	448	567	71	36
ACL 20k	30	35	5	3225	875	883	1484	97	56	30	35	5	3226	876	883	1486	99	56
ACL 30k	24	28	5	4590	970	978	2141	137	74	32	37	5	4593	973	977	2145	124	74
FW 10k	24	25	5	255	252	262	122	98	51	38	42	5	269	264	260	135	110	52
FW 20k	31	30	5	3255	2244	2247	1583	596	234	37	42	5	3270	2260	2249	1595	608	236
FW 30k	32	30	5	4713	2848	2851	3023	943	192	38	42	5	4731	2866	2854	3036	957	194
IPC 10k	22	27	5	1587	637	649	793	213	68	39	44	5	1604	656	655	809	229	71
IPC 20k	28	32	5	3142	712	723	1580	185	81	41	46	5	3163	732	732	1597	202	84
IPC 30k	28	32	5	4713	1072	1084	2247	354	192	41	46	5	4733	1093	1091	2265	372	197



(a) IP区域采用Trie-1

(b) IP区域采用Trie-2

(c) IP区域采用Trie-4

(d) IP域前缀转化为范围，建立RTree

图 3 BV, ABV和IABV算法内存占用量对比

19.4%，而FW规则和IPC规则内存占用量减少78.7%～91.8%。

综上，IABV算法内存占用量较ABV算法略有增加，但目前硬件工艺水平较高，算法内存增加量不会影响算法效率；4种结构中，将IP域前缀转化为范围然后建立范围树的方式占用的内存量最少，综合预处理时间以及内存访问次数两方面性能，将IP域前缀转化为范围处理包分类问题的方法相比Trie-1, Trie-2和Trie-4具有优势。

5　结束语

为加快ABV算法分类数据包的速度，本文将自适应哈希表与维度切分相结合提出了IABV算法。该算法建立了哈希表–规则库两级查找结构以快速分类数据包，提出了一种哈希表冲突处理机制解决哈希表中规则累积导致查找效率降低的问题，使用维度切分机制加快规则库查找速度，并在规则前缀域建立了不同的数据结构进行对比分析。实验结果表明IABV算法相比ABV算法具有更好的时间性能，空间性能方面略有下降。下一步工作将对算法的空间性能进行分析，降低算法空间复杂度。

参 考 文 献

[1] SHEN Tong, ZHANG Dafang, XIE Gaogang, *et al*. Optimizing multi-dimensional packet classification for multi-core systems[J]. *Journal of Computer Science and Technology*, 2018, 33(5): 1056–1071. doi: 10.1007/s11390-018-1873-9.

[2] BI Xiaan, LUO Xianhao, and SUN Qi. Branch tire packet classification algorithm based on single-linkage clustering[J]. *Mathematics and Computers in Simulation*, 2019, 155: 78–91. doi: 10.1016/j.matcom.2017.11.003.

[3] 亓亚烜, 李军. 高性能网包分类理论与算法综述[J]. 计算机学报, 2013, 36(2): 408–421. doi: 10.3724/SP.J.1016.2013.00408.
QI Yaxuan and LI Jun. Theoretical analysis and algorithm design of high-performance packet classification algorithms[J]. *Chinese Journal of Computers*, 2013, 36(2): 408–421. doi: 10.3724/SP.J.1016.2013.00408.

[4] 陈小雨, 陆月明. 基于多维空间动态划分与RFC的包分类改进算法[J]. 网络与信息安全学报, 2018, 4(3): 35–41. doi: 10.11959/j.issn.2096-109x.2018024.
CHEN Xiaoyu and LU Yueming. Improved packet classification algorithm based on multidimensional space dynamic division and RFC[J]. *Chinese Journal of Network and Information Security*, 2018, 4(3): 35–41. doi: 10.11959/j.issn.2096-109x.2018024.

[5] LAKSHMAN T V and STILIADIS D. High-speed policy-based packet forwarding using efficient multi-dimensional range matching[J]. *ACM SIGCOMM Computer Communication Review*, 1998, 28(14): 203–214. doi: 10.1145/285243.285283.

[6] 万云凯, 嵩天, 刘苗苗, 等. 流量自适应的多维度包分类方法研究[J]. 计算机学报, 2017, 40(7): 1543–1555. doi: 10.11897/SP.J.1016.2017.01543.
WAN Yunkai, SONG Tian, LIU Miaomiao, *et al*. A flow adaptive multi-dimensional packet classification algorithm[J]. *Chinese Journal of Computers*, 2017, 40(7): 1543–1555. doi: 10.11897/SP.J.1016.2017.01543.

[7] ZHOU Shijie, QU Y R, and PRASANNA V K. Multi-core implementation of decomposition-based packet classification algorithms[J]. *The Journal of Supercomputing*, 2014, 69(1): 34–42. doi: 10.1007/s11227-014-1205-y.

[8] ABBASI M, TAHOURI R, and RAFIEE M. Enhancing the performance of the aggregated bit vector algorithm in network packet classification using GPU[J]. *PeerJ Computer Science*, 2019, 5: e185. doi: 10.7717/peerj-cs.185.

[9] BABOESCU F and VARGHESE G. Scalable packet classification[J]. *IEEE/ACM Transactions on Networking*, 2005, 13(1): 2–14. doi: 10.1109/TNET.2004.842232.

[10] 贺亚威, 侯整风, 吴亮亮. 一种基于位向量流分类算法的改进[J]. 合肥工业大学学报: 自然科学版, 2015, 38(3): 331–335. doi: 10.3969/j.issn.1003-5060.2015.03.009.
HE Yawei, HOU Zhengfeng, and WU Liangliang. An improved flow classification algorithm based on bit vector[J]. *Journal of Hefei University of Technology: Natural Science*, 2015, 38(3): 331–335. doi: 10.3969/j.issn.1003-5060.2015.03.009.

[11] 李林. 防火墙规则集关键技术研究[D]. [博士论文], 电子科技大学, 2009.
LI Lin. Research on key techniques of firewall rule set[D]. [Ph.D. dissertation], University of Electronic Science and Technology of China, 2009.

[12] CAIDA. The CAIDA UCSD anonymized internet traces 2016[EB/OL]. http://www.caida.org/data/passive/passive_2016_dataset.xml, 2016.

[13] 颜天信, 王永纲, 石江涛, 等. 区域分割包分类算法的优化实现[J]. 通信学报, 2004, 25(6): 80–88. doi: 10.3321/j.issn:1000-436X.2004.06.011.
YAN Tianxin, WANG Yonggang, SHI Jiangtao, *et al*. Optimized implementation of regional partition algorithm for packet classification[J]. *Journal of China Institute of Communications*, 2004, 25(6): 80–88. doi: 10.3321/j.issn:1000-436X.2004.06.011.

[14] TAYLOR D E and TURNER J S. ClassBench: A packet classification benchmark[C]. Proceedings of the IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies, Miami, USA, 2005: 2068–2079. doi: 10.1109/INFCOM.2005.1498483.

[15] 孙鹏浩, 兰巨龙, 陆肖元, 等. 一种基于匹配域裁剪的包分类规则集压缩方法[J]. 电子与信息学报, 2017, 39(5): 1185–1192. doi: 10.11999/JEIT160740.
SUN Penghao, LAN Julong, LU Xiaoyuan, *et al*. Field-trimming compression model for rule set of packet classification[J]. *Journal of Electronics & Information Technology*, 2017, 39(5): 1185–1192. doi: 10.11999/JEIT160740.

张　斌：男，1969年生，教授、博士生导师，研究方向为网络空间安全.

吴浩明：男，1995年生，硕士生，主要研究方向为网络流量检测.