



多域网分组分类算法研究^{*}

刘寒冰, 杨亚丽, 胡爱娜

(黄河科技学院信息工程学院 郑州 450063)

摘要: 随着网络业务的不断增长, 网分组分类技术成为下一代互联网的研究热点。多域网分组分类算法作为网分组分类的核心技术, 具有重要的研究价值。介绍了网分组分类算法的理论基础; 对典型的网分组分类算法进行了介绍和比较分析, 指出了各自存在的优缺点, 并针对性地提出了改进措施; 最后对多域网分组分类算法加以总结, 探讨了新的研究方向。

关键词: 多域网分组; 五元组; 载荷

doi: 10.3969/j.issn.1000-0801.2014.12.015

Research on Multi-Dimensional Packet Classification Algorithm

Liu Hanbing, Yang Yali, Hu Aina

(School of Information Engineering, Huanghe College of Science and Technology, Zhengzhou 450063, China)

Abstract: With the increase of network services, multi-dimensional packet classification technology has become a core technology for network devices in the next generation internet. Firstly, the theoretical foundation of packet classification algorithm was introduced. Secondly, the typical packet classification algorithms were introduced and analyzed, and the improvement measures were proposed. Finally, the multi-dimensional packet classification algorithm was proposed and the future work was discussed.

Key words: multi-dimensional packet, 5-tuple, payload

1 引言

随着网络技术的发展和网络业务的扩张, 网分组分类技术作为支撑业务融合、网络安全、流量监控等网络功能的核心技术, 能够有效推动网络融合的业务发展及网络基础设施建设, 提升网络安全设备性能, 在业务融合、网络安全及网络监控等诸多领域都有广泛应用。因此, 研究和分析网分组分类问题, 设计并实现高性能的实用网分组分类算法, 具有极为重要的科学价值和应用场景。

多域网分组分类是指根据网分组中的多域信息, 按照给定的规则集合对网分组进行分类。由于多域网分组分类

能够依据网分组信息对网络流量进行细粒度的划分, 已在路由器、安全网关及流量控制系统等各类网络设备中得到了广泛应用^[1,2], 并成为下一代互联网发展和演进中的研究热点^[3-6]。

网分组分类算法作为网分组分类的核心技术, 自 20 世纪 90 年代以来, 受到学术界的高度重视, 出现了很多研究成果。最早的多域网分组分类算法是根据五元组域规则进行匹配实现分类的, 近年来, 根据载荷内容进行网分组分类的算法受到了越来越多的关注。然而, 对算法研究成果的总结却很少, 仅有 21 世纪初的若干篇综述文章, 最近的研究成果则缺乏系统的分析和总结。因此, 为了推

^{*} 河南省科技发展计划基金资助项目 (No. 122102310412, No. 112102310582), 郑州市重点实验室资助项目 (No. 121PYFZX177)



动多域网分组分类算法的研究和发展,本文从两个方面对近年来的多域网分组分类算法进行了总结和对比分析,具体介绍如下。

- 总结了网分组分类算法的理论基础。对算法的分类规则和评价指标进行归纳,并从理论和设计两个层面对现有算法进行总结分析,指出存在的问题,为新算法的提出和已有算法的优化提供了理论依据。
- 介绍并分析了现有多域网分组分类算法。对算法的设计思想、性能进行了深入的分析和比较,指出每种算法的优缺点,并结合自身研究经验,提出改进措施和新的研究方向。

2 网分组分类算法的理论基础

本节首先将网分组分类问题归结为几何空间中的点定位问题,然后介绍了两类网分组分类规则,最后从理论和设计层面分析了网分组分类算法的性能评估标准。

2.1 网分组分类算法的分类规则

网分组中的多域信息构成多域空间,为分类而构建的规则作为此空间中的若干个点,因此网分组分类算法本质上是多域网分组空间中的规则点定位问题。

(1) 基于五元组分类

分组头的五元组构成5个域,并且每个域的长度固定,包括源/目的IP地址域(32 bit)、源/目的传输层端口域(16 bit)以及传输层协议域(8 bit)。基于五元组分类是指根据5个域中的内容是否满足规则要求对网分组进行分类。每一条规则都可以相应地投影到这5个域中,每个投影都对应相应域中所要匹配的规则。如果5个域中的网分组信息与规则在5个域中的投影都匹配,则说明此网分组满足相应规则,需要执行规则对应的操作,如接受转发、拒绝转发、重置连接或丢弃网分组等。

(2) 基于载荷分类

传统的网分组分类系统依据多域网分组的分组头进行分类,而现代网络设备越来越注重基于网分组载荷的网分组分类,从而改善数据传输,并提高网络安全。基于载荷的网分组分类是实现深度检测的核心技术,广泛应用于入侵防御、协议识别、内容过滤等网络设备。网分组内容除去分组头后,其余数据部分称为载荷域,基于网分组载荷的网分组分类是指将网分组载荷中的字节流与分类规则一一匹配,并依据匹配结果对网分组进行分类。由于网分组载荷的长度不定,无法按照五元组域匹配的方法实现,载

荷中每一个信息都相当于一个域,因此,依据网分组载荷的网分组分类是多域网分组分类的拓展,即从多域向无穷域的拓展。基于载荷的分类规则通常使用正则表达式以及自动状态机(DFA)实现。

2.2 网分组分类算法的评价指标

多域网分组分类算法的基本思想是对给定规则集合进行预处理,生成高效的查找数据结构,有效提高网分组分类系统的性能。算法性能通过预处理时间、吞吐率、内存访问次数、内存占用等指标体现。

从算法理论层面来看,以美国斯坦福大学、美国加州大学圣地亚哥分校、清华大学以及微软研究院为主导的网分组分类算法研究,均以 Overmars 等人提出的基于回溯查找的多域区间树算法作为网分组分类问题的理论最优解^[7]。由于该算法的时间复杂度不能满足大多数网分组分类系统的性能要求,工业界流行的网分组分类技术并未普遍应用这些算法,而大多采用了不使用回溯查找的方法,以提高处理效率。

从算法设计层面来看,现有网分组分类算法虽然能够充分发掘实际网分组分类规则的特性,利用启发式方法构建优化的查找数据结构,但往往偏重于单一性能指标的提高。如 RFC 算法^[8]具有极高的分类速率,但需要大量的存储空间;与之相反,EffiCut 算法^[9]虽然存储空间得到了极大优化,但存储访问次数却增加了数倍之多。

3 基于五元组分类算法

最早基于五元组域的分类算法是由 Srinivasan 提出的 Cross-producting 算法^[10]。基于 Cross-producting 算法,学术界提出了一系列的网分组分类算法,其中最典型的是 RFC 算法^[8]、HyperSplit 算法^[11]和 HSM 算法^[12]。作为算法改进和创新的依据,本文将对这几类典型算法进行介绍和分析。

3.1 Cross-producting 算法

Cross-producting 算法是最早出现的多域网分组分类算法之一,其主要思想是采用最长前缀匹配方法进行各域上的规则投影区间查找,并使用求交表完成空间求交。

从数据结构角度分析,Cross-producting 算法首先使用字典树结构进行最长前缀匹配,并存储每个域的不同前缀及其对应的前缀编号。然后使用一个 d 维的求交表进行空间求交, d 表示域的个数, n 表示规则个数。其中, d 维求交表各个维度的取值大小由该维度上不同前缀的个数决定,而每个表项存储内容为优先级最高的规则序号。

Cross-producting 算法将规则投影区间查找和最长前缀查找结合,成为最早的网分组分类算法之一。Cross-producting 算法为其后续算法 RFC、HSM 等提供了设计思路。然而,Cross-producting 算法的缺点也十分明显。首先,Cross-producting 算法要求对规则集合进行范围前缀转化,这种转化会大幅降低算法的空间性能;其次,由于 Cross-producting 算法的时间复杂度为 $O(d \times W)$ (W 表示树的深度),对于典型的五元组分类需要上百次的内存访问(典型的五元组网分组组头为 104 bit),所以网分组分类速率不能得到有效保障;最后,单一的 d 维求交表无法利用启发式方法消除分类规则的结构冗余。

3.2 RFC 算法

为解决 Cross-producting 算法中分类速率低和内存使用过高的问题,Gupta 等人提出了 RFC 算法。RFC 算法的基本思想是:利用数组结构存储各域的投影区间,将每个维度上 $O(W)$ 的最长前缀查找时间提高为 $O(1)$,同时采用多级求交表进行迭代求交,逐级消除空间冗余。

RFC 算法的数据结构为多级表结构,如图 1 所示,这些表被称为“chunk”,表结构的级数(phase)以及每一级中 chunk 的个数都是预先设定的,图 1 给出了一种可能的结构。这些表的建立分多步进行:在第一级(phase 0)中,对网分组组头五元组的各个域进行规则投影区间划分,以地址或端口为索引,为各个维度建立对应的表,每个表项存放对应的子空间序号;在下一级中,每一个表都由前一个 phase 的若干个表求交生成,索引由各个域的子空间序号计算得出,每个表项存储当前搜索空间划分方法的子空间序号;最后一级求交表中,每个表项存储优先级最高的匹配规则序号。

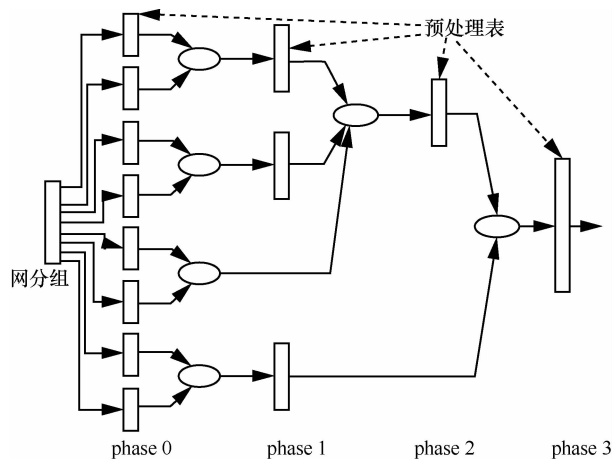


图 1 RFC 算法结构示例

RFC 算法的优点是:利用数组查找避免了每个域上最长前缀的查找,将分类速率提高到了 $O(d)$;通过多级映射,消除各级表项中的空间冗余。RFC 算法的不足之处包括:表结构级数和每级 chunk 个数都是提前设定的,因此当规则集合变化时,这种设定可能不是最优;利用数组进行区间查找,无法实现对 32 bit IPv4 地址的索引,因此 RFC 算法将 32 bit 的 IP 地址域拆分为两个 16 bit 的维度,建立两张 chunk 表,这实际上增加了下一级 chunk 的表项数,尤其对于 IPv6 中的 128 bit 的 IP 地址,这种拆分方式更不适用。

3.3 HSM 算法

Xu 提出的 HSM 算法改善了 RFC 算法中存在的一些问题。HSM 算法的基本思想是利用二分查找法进行各域的区间查找,使用两级映射表进行空间求交。HSM 算法能够进一步降低 RFC 算法的内存空间,同时能够支持任意位数的多域网分组分类。图 2 为一个四域的 HSM 算法示例,HSM 算法的查找过程分为 3 个阶段,具体介绍如下。

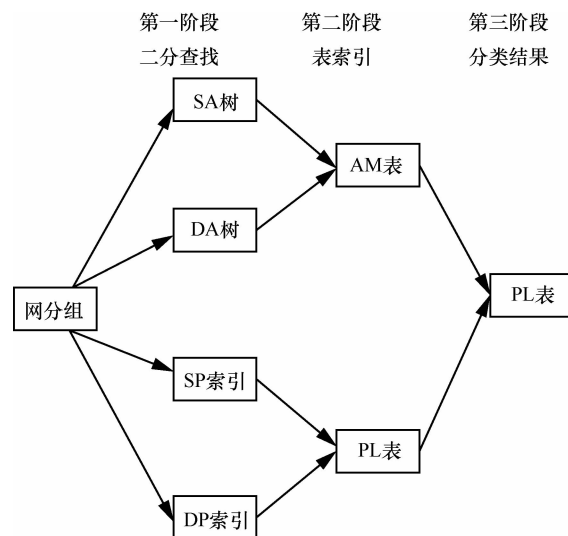


图 2 HSM 算法原理

第一阶段:根据规则集合在源 IP 地址(SA)、目标 IP 地址(DA)、源端口(SP)和目标端口(DP)4 个域上进行投影区间查找。为了加快查找过程,HSM 算法对于 32 bit 的 IP 地址使用平衡二分查找树查找,而对于 16 bit 的端口号使用 RFC 算法中的数组索引查找。

第二阶段:将 SA 和 DA 的区间编号作为索引,对地址映射表(address mapping table,AMT)进行查找。AMT 是 SA 和 DA 两个域的空间求交表,每个表项为指定地址组编号(address group number,AGN)。第二阶段将 SP 和 DP 的区



间编号同时作为索引,对端口映射表(port mapping table, PMT)进行查找。PMT 是 SP 和 DP 两个域的空间求交表,每个表项为指定端口组编号(port group number, PGN)。

第三阶段:将 AGN 和 PGN 作为索引,对策略查找表(policy lookup table, PLT)进行查找。PLT 为 AMT 和 PMT 两个表中独立元素个数的求交表,每个表项存储最后的匹配规则编号。

由于使用了二分查找代替数组索引, HSM 算法的时间复杂度为 $O(d \times \log n)$ 。虽然时间复杂度较 RFC 算法有所提高,但由于实际应用中,源 IP 域的区间数极小,同时目标端口和源端口域又采用了数组索引,因此 HSM 算法的分类速率实际仅由目标地址域决定。HSM 算法和 RFC 算法一样,被认为是分类速率最快的一类算法。

由于 HSM 算法在空间求交过程中依然采用了确定性的求交方法,因此并未从根本上解决 RFC 算法存储空间过大的问题。此外,无论 RFC 算法还是 HSM 算法在处理大规模规则集合时,预处理时间都很长。

3.4 HyperSplit 算法

亓亚焯等人提出的 HyperSplit 算法^[11]改善了 HSM 算

法中存在的一些问题,在保持查找算法高效分类能力的同时,大幅度降低了算法的内存使用。

HyperSplit 算法采用了没有回溯的二分查找,因此时间复杂度为 $O(d \times \log n)$,与 HSM 算法一致。在构建二分查找树时,不同于 HSM 算法在每个域上进行独立的、全规则集合投影区间的二分查找,HyperSplit 算法采用了启发式的区间查找策略。HyperSplit 算法每次选取最优的二分点(最优选取需要单独算法设计),并以过该点且垂直于坐标轴的平面将搜索空间二分。由于每次划分后规则集合也分为两个子集,因此下一次区间二分仅对两个规则子集分别处理即可,即实际区间数逐级降低。

为便于说明 HyperSplit 算法的核心思想,图 3 和图 4 给出了 HSM、HyperSplit 两种算法的分解策略实例。可以看出, HSM 算法中 r_2, r_3, r_5 均出现了复制,而 HyperSplit 算法并未复制规则,因此节省了存储空间。从搜索过程来看, HSM 算法分别查找两个树结构最多需要 4 次二分查找,而 HyperSplit 算法最多只需 3 次查找。

虽然理论上 HyperSplit 算法和 HSM 算法具有相同的时间复杂度,但 HyperSplit 算法的实际内存访问次数却明

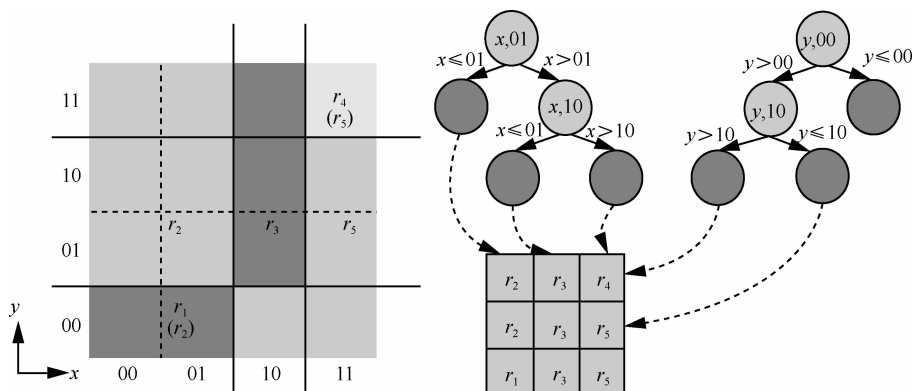


图 3 HSM 算法示例

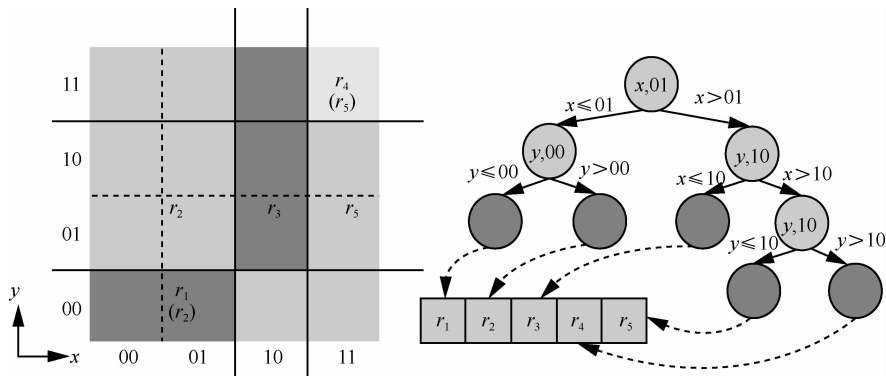


图 4 HyperSplit 算法示例

显少于 HSM 算法。这是由于 HyperSplit 算法每次二分后, 仅对规则子集而非规则全集的投影区间继续二分。

HyperSplit 算法在分类速率以及内存使用方面都优于 HSM、RFC 等算法, 其缺点是系统吞吐率低以及预处理时间长。通过将其部署于多核处理器, HyperSplit 算法的吞吐率约为 6.4 Gbit/s, 目前网络中所需要的吞吐率很多都要求大于 40 Gbit/s; 在预处理时间上, 对于较大型的规则集合, HyperSplit 算法预处理时间需要几分钟或者几十分钟, 不能满足网络快速处理的需求。

3.5 对比分析

上述 4 种算法 Cross-producting、RFC、HSM 和 HyperSplit 在数据结构方面, RFC 算法使用数组结构存储规则, 预处理速度以及查找速度都可以达到很好的性能, 但是不能适应规则的变化性, 只能应用于特定的少量规则情形; 其余 3 种算法均使用树型结构存储规则, 预处理时间较长, 在查找速度以及内存访问次数方面不能完全满足实际网络需求。

在内存使用方面, 当使用 FW1-10K 规则集时, HyperSplit 算法占用的内存比 HSM 算法小 1 个数量级; 在预处理时间方面, 对于 FW1-10K 规则集, HyperSplit 算法的预处理时间为 10 min, 虽然小于 HSM 算法, 但仍然不能满足实际需求; 在系统吞吐率方面, 通过部署 16 核处理器, HSM 算法能达到 3 Gbit/s 左右的最大吞吐率, HyperSplit 算法能达到 6.4 Gbit/s 的最大吞吐率。

HyperSplit 算法在目前的同类算法中, 性能优势比较明显, 内存使用、预处理时间都领先其他方法 1~2 个数量级。FW1-10K 规则集下的性能评测数据见表 1。

表 1 FW1-10K 规则集下的性能评测数据

算法	内存使用	预处理时间	吞吐率
HSM	>4 GB	>24 h	3 Gbit/s
HyperSplit	753 MB	10 min	6.4 Gbit/s

提高查找速度、预处理时间的一种方法是使用专用存储器 TCAM (ternary content addressable memory, 三态内容寻址存储器) 替代树型结构。Hua N 提出的 VS-DFA 算法^[13]使用了 TCAM 作为存储结构。

4 基于载荷分类算法

最早的基于载荷分类算法是由 Thompson 等人提出的 DFA 算法^[14]。随后, Kumar 等人对 DFA 算法进行改进, 提出了 D²FA 算法^[15], 卞亚焯等人提出的 FEACAN 算法^[16]也

对 DFA 算法做了改进。Nan H 等人利用模式匹配的方法提出了 VS-DFA 算法^[13]。下面分别对这几种算法进行介绍。

4.1 D²FA 算法

对同一种规则描述, 既可以用有穷自动机描述, 也可以用正则表达式描述, 所以, 对于使用正则表达式进行规则匹配的网分组分类, 通常使用有穷自动机实现。不确定有穷自动机 (NFA) 与确定性有穷自动机 (DFA) 的根本区别是它们的跳转函数不同。DFA 对每一个可能的输入只有一个状态的跳转; NFA 则对每一个可能的输入有多个状态跳转, 在接收到输入时, 从这多个状态跳转中非确定地选择一个。

将正则表达式转换成 DFA 的基本方法是先把正则表达式转换成 NFA; 然后使用子集构造 (subset construction, SC) 法将 NFA 转换成 DFA; 最后再将 DFA 最小化。对于任意的正则表达式, 都可以找到一个与之等价的具有最少状态数的 DFA。

Kumar 等人注意到 DFA 中的许多状态有相似的相邻状态跳转, 于是根据模式匹配中的 AC 算法, 提出了默认跳转 (default transition, DT) 的概念, 并基于此提出了一种优化的 DFA 结构——D²FA。D²FA 通过默认路径和默认跳转, 实现 DFA 表的压缩。每一条默认路径上的状态仅存储不同的跳转, 而默认跳转可以由其同路径上的父状态获得。在不控制默认路径深度的前提下, D²FA 算法对由常见正则表达式规则集合生成的 DFA 有 95% 的压缩率。

图 5 和图 6 给出了 4 个状态 a、b、c、d, 3 个正则表达式 a+, b+c, cxd+ 的 DFA 以及与 DFA 等价的 D²FA 状态转移图。

从图 5 和图 6 中可以看出, D²FA 以默认跳转取代了大量重复的标号跳转, 取得了非常高的空间压缩比。然而,

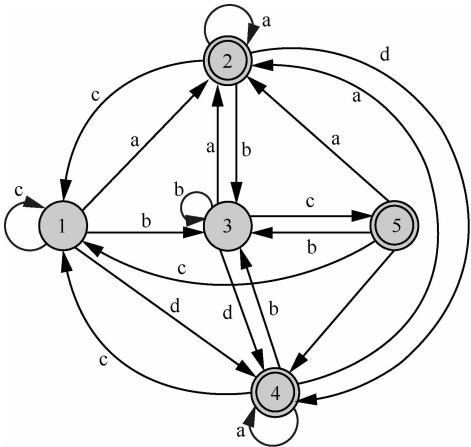
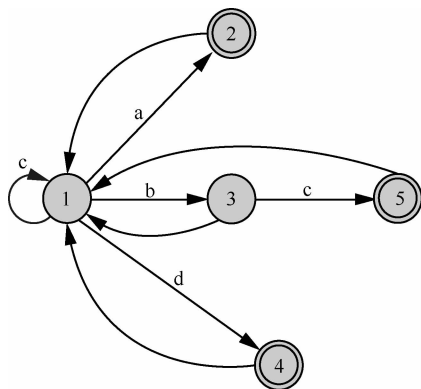


图 5 对应正则表达式 a+、b+c、cxd+ 的 DFA

图6 对应正则表达式 $a^+b^+c^+d^+$ 的 DFA

DFA 实际上是以增加处理每个字符时经过的状态数或跳转数为代价的,这在一定程度上降低了 DFA 的时间性能。

Becchi 等人指出了 DFA 算法存在的 3 个问题,具体介绍如下。

- DFA 算法性能与最长默认路径直接相关,对其长度限制的参数是需要经过大量实验之后才能确定的,因为对于不同的规则集,这个参数的最优值往往不同。
- DFA 的构造算法构造出的数据结构,最坏情况下最长默认路径每一次都会被遍历。
- DFA 的构造算法需要多次扫描全局状态,因此实现效率低。

4.2 FEACAN 算法

FEACAN 算法^[6]的核心思想是在保证 DFA 查找速率的同时,尽可能压缩 DFA 跳转矩阵的大小。参考文献[16]说明了状态跳转矩阵是非常稀疏的,这为进一步压缩跳转矩阵提供了理论依据。

FEACAN 算法通过 3 个步骤实现状态跳转矩阵的状

态内比特串压缩和状态间两级聚类压缩。

(1) 状态内比特串压缩

FEACAN 算法使用 bitmap 方法实现状态内的比特串压缩,如图 7 所示,某个状态的跳转向量经过压缩后,只需要保留互不相同的若干个状态以及 1 个比特串。

(2) 状态间压缩——第一级状态聚类

对于压缩后的状态,FEACAN 算法使用第一级聚类把相同的比特串对应的状态排列在一起,这样可以减少比特串的存储数量(即只需存储不同的比特串)。如图 8 所示,在完成第一级聚类(总共 K 个组)后,只留下了 K 个不同的比特串,且它们每一个都对应一个组中所有的状态。

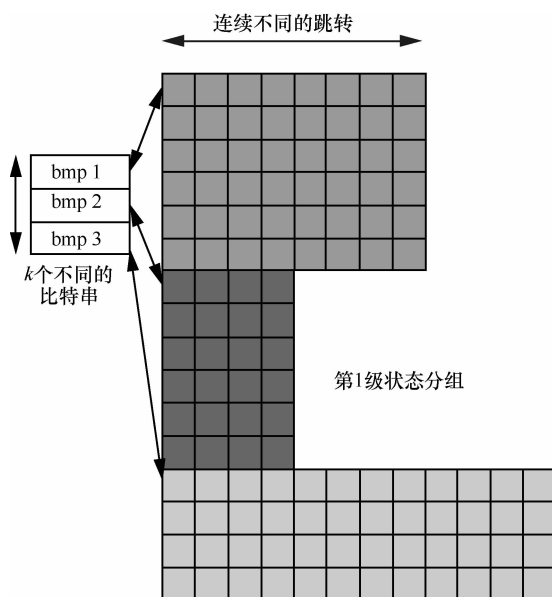


图8 第一级状态聚类

(3) 状态间压缩——第二级状态聚类

对于经过第一级聚类压缩后的状态,FEACAN 算法将

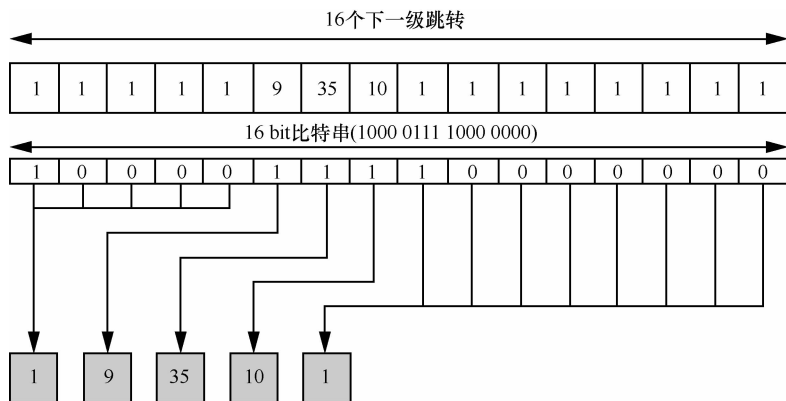


图7 使用 bitmap 压缩一个状态

相似的状态排列在一起,判断相似的标准是两个状态转移向量之间共享相同的比特串,并且多于 80%的跳转是相互关联的。

在两个层的聚集之后,FEACAN 算法选每组中的第一个状态作为首领状态,并把其他状态作为成员状态。算法通过消除成员状态中与首领状态相同的跳转进行状态间压缩。

通过首领状态的选取以及相同状态的删除,得到最终的跳转矩阵压缩结果,如图 9 所示。

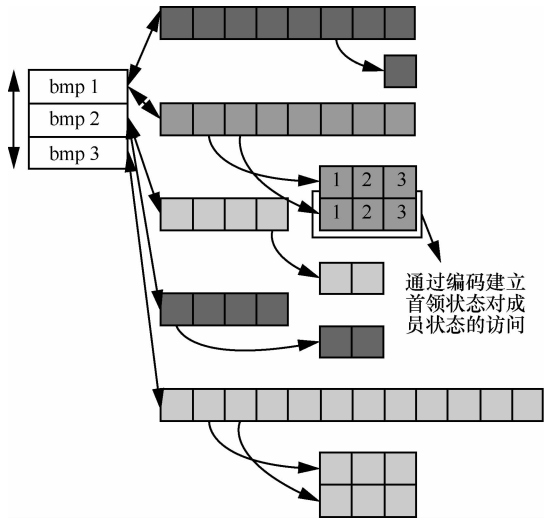


图 9 最终压缩结果

通过这 3 步的压缩,FEACAN 算法比原始的比特串压缩方法减少最高 80%的内存使用,提高了内存使用效率,极大地改善了性能。同时,FEACAN 算法也存在一定的缺陷:在压缩过程中,此算法加入了两级聚类压缩、一级 bitmap 压缩,提高了运算复杂度,降低了预处理速度;对于一些复杂的正则表达式规则,原始 DFA 的状态数会呈爆炸式增长,这种情况下将无法直接使用 DFA 或 FEACAN 等算法对 DFA 进行压缩。

4.3 VS-DFA 算法

FEACAN 算法是针对正则表达式表示的规则集合,实现基于载荷的分组分类,另外一类规则集合是以模式匹配的形式出现的。Hua N 等人提出的 VS-DFA 算法就是通过模式匹配实现分组分类的一种算法^[13]。

传统的模式匹配方法是按照单字符建立 DFA 的,查找效率较低,并且占用大量的内存空间。VS-DFA 算法的核心思想是对规则集合以及载荷内容做确定性的划分,依据划分结果建立 DFA,每个 DFA 中的状态都超过 1 个字符,

提高了查找效率。

划分方法采用 Wnnowing 算法^[17]对长为 L 的字符串中每 k 个字符做散列计算,得到 $value$ 值,对于各个 $value$,以 w 宽的窗口滑动,选取最小 $value$ 值所在的字符位置作为一个划分,直到字符串都处理结束。如图 10 所示,以 $k=2$ 、 $w=3$ 划分,得到 1 条关键词的划分结果。

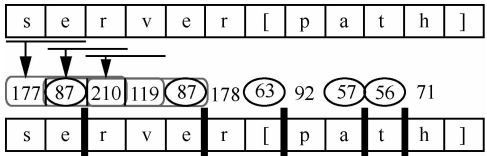


图 10 Wnnowing 算法的划分结果

每条规则划分的结果都可以分为 headblock、coreblock 和 tailblock,如图 11 所示。s6 没有 coreblock,称作 coreless string;s7 自身就是一个 block(块),不可分,称作 indivisible string。

	head string	core string	tail string
s1:ridiculous	s1: r	id ic ulo u	s
s2:authenticate	s2: auth	ent ica	te
s3:identical	s3: id	ent ica	l
s4:confident	s4: conf	id	ent
s5:confidential	s5: conf	id ent	ial
s6:entire	s6: ent	(empty-core)	ire
s7:set	s7: ---	(indivisible)	---

图 11 划分结果(3 种 block)

如果对载荷数据按照同样的方法划分,当数据中包含规则中的字符串时,Wnnowing 算法的划分结果中,coreblock 部分与规则划分的 coreblock 部分完全相同,只是 headblock 和 tailblock 有所差异。Wnnowing 算法对 coreless string 和 indivisible string 有特殊处理,以保证不发生漏检。

根据上述特性,VS-DFA 算法的操作过程为:首先,输入数据流被分割为许多个 block;然后,这些 block 都进入一个 FIFO;最后,通过 VS-DFA 状态机查找搜索,输出匹配结果,如图 12 所示。

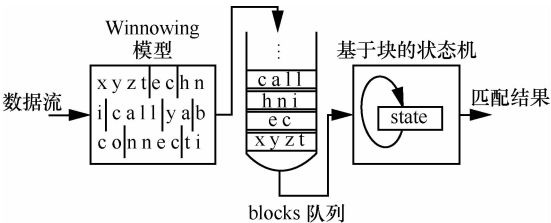


图 12 VS-DFA 算法模型



VS-DFA 算法针对的规则集合为模式匹配形式,目前还不能应用于正则表达式匹配,因为正则表达式中包含通配符等情形,不适合做 block 划分。同时,单个 block 的大小也受到一定的限制,block 过大或过小都会导致性能下降。如何在不影响系统性能的同时取得更大的 block,可以作为下一步的研究目标。

4.4 比较分析

DFA 算法作为早期的载荷匹配算法,为之后的算法提供了基础。D²FA 算法以及 FEACAN 算法都是在 DFA 算法的基础上针对正则表达式匹配做的改进,FEACAN 算法的改进效果要优于 D²FA 算法,大幅度降低了内存使用,但是提高了预处理复杂度。VS-DFA 算法为了提高系统吞吐率、处理速度,将 DFA 算法中单字符作为一个状态,改进为多字符 block 整体作为一个状态,极大地提高了吞吐率,但不能处理正则表达式的情形。

对于内存使用以及预处理时间,在 snortAll 规则集下,根据各个算法的复杂度可以得到表 2 中的数据;对于系统吞吐率,在现场可编程门阵列(FPGA)中利用 Xilinx 芯片实现 FEACAN 算法^[15],其余两种算法没有进行吞吐率测试。具体性能见表 2(snortAll 规则集)。

表 2 snortAll 规则集下的性能评测结果

算法	内存使用	预处理时间	吞吐率
D ² FA	6 MB	20 min	
FEACAN	<2 MB	10 s	9.6 Gbit/s
VS-DFA	<2 MB	<1 s	

下阶段研究中,如果能够对 VS-DFA 算法进行改进,以适应正则表达式匹配,并结合 FEACAN 算法中的压缩方法,将极大提高算法的使用范围,并且可能成倍地增加系统吞吐率,降低预处理时间;如果能够改善划分 block 的方法,还会对性能提升有较大的帮助。

5 结束语

在基于五元组域的分类算法中,HyperSplit 算法相对较优,但其不足之处在于,树型结构存储规则导致预处理时间较长、处理速度较慢,无法达到较高的系统吞吐率,因此,如何通过改进算法加快预处理速度,并通过并行处理或者其他技术提高吞吐率是基于五元组域分类算法的一个研究方向。在载荷域分类算法中,FEACAN 算法和 VS-DFA 算法各有优势,FEACAN 算法重点解决

了跳转矩阵的高效压缩问题;VS-DFA 算法以 block 取代单字符作为 DFA 的状态单元,提高了吞吐率,改善了性能,因此,如何将两种算法取长补短是一个值得研究的方向。

传统的五元组域匹配分组分类方法在后续的应用中,依然会发挥很大的作用,如路由转发、防火墙等领域,改进方向应该致力于提升系统吞吐率、降低内存使用、提高处理速率等;而基于载荷匹配的方法正处于发展阶段,越来越多的应用也会需要这种方法,因此应给予基于载荷的匹配方法更多关注。

参考文献

- 1 Varghese G. Network Algorithmics: an Interdisciplinary Approach to Designing Fast Network Devices. San Francisco: Morgan Kaufmann Publishers, 2005
- 2 Chao J, Liu B. High Performance Switches and Routers. New Jersey: Wiley, 2007
- 3 徐恪,吴建平,徐明伟. 高等计算机网络——体系结构、协议机制、算法设计与路由技术(第二版).北京:机械工业出版社,2009
- 4 Joseph D, Tavakoli A, Stoica I. A policy aware switching layer for data centers. Proceedings of the ACM SIGCOMM, Seattle, USA, 2008:51~62
- 5 Koponen T, Casado M, Gude N, *et al.* Onix: a distributed control platform for large-scale production networks. Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, Vancouver, Canada, 2010:351~364
- 6 Opa L, Egi N, Ratnasamy S, *et al.* Building extensible networks with rule-based forwarding. Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, Vancouver, Canada, 2010:379~392
- 7 Overmars M, Stappen A. Range searching and point location among fat objects. Journal of Algorithms, 1996, 21(3):629~656
- 8 Gupta P, McKeown N. Packet classification on multiple fields. Proceedings of ACM SIGCOMM, Cambridge, Massachusetts, USA, 1999:147~160
- 9 Vamanan B, Voskuilen G, Vijaykumar T. EffiCuts: optimizing packet classification for memory and throughput. Proceedings of ACM SIGCOMM, New Delhi, India, 2010: 207~218
- 10 Srinivasan V, Varghese G, Suri S, *et al.* Fast and scalable layer four switching. Proceedings of ACM SIGCOMM, Vancouver, Canada, 1998:191~202
- 11 亓亚焯,李军. 多域网分组分类算法研究. 清华大学博士学位论文,2011