面向网络流的自适应正则表达式分组匹配算法

杜文超，陈庶樵，胡宇翔
（国家数字交换系统工程技术研究中心，450002，郑州）

摘要：针对当前的多正则表达式匹配算法占用较大的系统资源，且吞吐量较低的问题，在分析典型的正则表达式匹配算法的基础上，提出了一种自适应的多正则表达式分组匹配算法. 该算法通过对正则表达式进行高效分组，将相互之间存在交叠且容易引起状态数指数增长的表达式相互隔离；将每个分组构造为一个确定性有限自动机(DFA)，按匹配概率大小建立伸展树进行调度. 仿真结果表明，该算法不仅大大节省了存储空间，而且吞吐量提高了大约3倍.

关键词：深度包检测；正则表达式；分组；有限自动机；伸展树
中图分类号：TP393 **文献标志码**：A **文章编号**：0253-987X(2012)08-0049-05

An Adaptive Matching Algorithm for Regular Expression Grouping in Network Traffic

DU Wenchao, CHEN Shuqiao, HU Yuxiang
(National Digital Switching System Engineering & Technological R&D Center, Zhengzhou 450002, China)

Abstract: In view of current multiple regular expressions matching algorithms consume too large system resource, but provide low throughput, a new regular expression matching algorithm for network context is proposed through analyzing typical regular expression matching algorithms. The algorithm separates expressions that have interaction and could cause exponential increase in the number of states, and assigns them into different groups. Then a DFA is constructed for each group, and these DFAs are placed on the splay tree to schedule by their matching probabilities. Simulation results show that the proposed algorithm not only could save more memory space, but also could provide about three times higher throughput.

Keywords: deep packet inspection; regular expression; group; finite automata; splay tree

面向网络流的多正则表达式匹配被广泛应用于网络流数据采集与处理、网络安全检测等领域. 传统的检测系统多数采用基于字符串的多模式匹配算法，如Aho-Corasick算法[1]. 随着检测的内容日渐复杂，正则表达式由于具有强大的表达能力与灵活性，开始逐渐代替精确字符串作为主要的报文特征描述方式. 当前，很多内容检测引擎已经包含有大量的正则表达式，如Snort、Bro等入侵检测系统.

正则表达式匹配的原理是将正则表达式构造成有限状态机(FSM)，再用FSM对数据内容进行扫描，它具有两种实现方式：确定性有限自动机(DFA)和非确定性有限自动机(NFA). NFA可能有多个活跃状态，匹配效率较低；相比之下，DFA具有较少的访存次数，但当规则数量增多时，状态数量急剧增长，容易引起状态空间爆炸，当前的硬件无法满足如此大的内存需求.

正则表达式匹配产生状态空间爆炸主要表现为以下两个方面：①当单个正则表达式中包含有长度限定的通配符时，如表达式 $.*a+.\{i\}cd$ 中 $a+$ 与 $.\{i\}$ 之间存在交叠，DFA需要记录 c 之前的最后 i

收稿日期：2011-12-14. 作者简介：杜文超(1984—)，男，硕士生；陈庶樵(通信作者)，男，教授. 基金项目：国家重点基础研究发展计划资助项目(2012CB315901)；国家科技支撑计划资助项目(2011BAH19B01).
网络出版时间：2012-05-11 网络出版地址：http://www.cnki.net/kcms/detail/61.1069.T.20120511.1145.004.html

个字符出现的所有可能,需要 $O(2^i)$ 个状态,当规则库中含有此种表达式数量较多时,容易引起状态空间爆炸,文献[2]指出 snort 规则库中超过 10%的表达式含有长度限定通配符;②当表达式集合中含有无限长度限定通配符,且当所有的表达式建立一个组合 DFA 时,如 r1:. $*a$. $*b$,组合 DFA 需要记录与前缀中含有字符 a 的哪个子集进行了匹配,当表达式集合中含有 m 个表达式时,需要 2^m 个状态,此时也容易引起状态空间爆炸.

1 相关工作

正则表达式匹配是当今网络管控领域的一个研究热点.针对正则表达式匹配时存在的状态空间爆炸问题,近年来研究者提出了多种正则表达式匹配算法,如 Becchi 等[3]提出基于状态合并的 DFA,Ficara 等[4]提出两级转移压缩 δ^2FA,Kong 等[5]提出基于多字母表压缩 DFA,以及在这些算法基础上的多种改进算法,都在不同程度上对 DFA 的存储空间进行了压缩,取得了较好的压缩效果,但多数以内存带宽和匹配速率为代价来达到存储空间优化的目的,仍存在着匹配效率低下的缺点,同时在处理动态变化的网络流时则无能为力.

通常,为了得到较快的匹配速率,最直接的方法就是把所有的表达式构建为一个 DFA,但这会造成 DFA 的状态数量十分庞大,现有的硬件资源无法满足如此高的存储空间需求.因此,最好的办法就是将表达式集合进行分组,对每个分组单独创建一个 DFA. Yu 等[2]提出两条重写规则,但适用性有限;同时又提出启发式分组算法,但需要对规则集中任何两个表达式之间的交叠状况进行计算评估,对于包含数万条规则的规则库显然是不现实的,需要较长的处理时间,严重影响了匹配速率;Jiang 等[6]提出一种新型的分组算法后建立 P-DFA,通过组合 DFA 一系列的插入、删除操作对整体状态机的影响,来估算表达式之间的关系后进行分组,然而这仍然不是对分组与计算复杂度之间一个很好的折中方案,况且此种方法只对其进行分组,并没有提出相应的调度方法;Xu 等[7]提出一种面向网络流的正则表达式匹配算法,但该算法对每个表达式逐个建立对应的 DFA,仍然不适合包含有成千上万条规则的情况,另外 DFA 数量较大时,对于每个输入字符需要逐个检查每个 DFA,计算复杂度较大;Ficara 等[8]提出了一种数据流抽样方法来对网络流进行检测,采用两级处理,首先对数据流进行试抽样匹配,若匹配则做进一步验证,但该算法只注重吞吐量的提高,忽视了存储空间的消耗问题.

由此可见,传统的算法忽略了网络流的上下文,造成吞吐量有限、效率低下.本文从网络流的上下文出发,在预处理阶段对表达式进行合理分组,然后对各个分组建立相应的 DFA,最后将 DFA 按匹配概率大小构建成伸展树,使匹配概率大的位于树的根部附近,而匹配概率小的则离根部较远,动态调整各个 DFA 的位置,以适应网络流的变化.

2 算法介绍

算法主要分为两个步骤:正则表达式集合分组和根据匹配概率大小建立伸展树 splay tree[9]进行调度.基本流程如图 1 所示,首先提取网络应用协议特征并用正则表达式表达,然后将表达式采取分组算法进行分组,对每个分组创建 NFA 后转化为 DFA,最后创建 splay tree,按照匹配概率大小将这些 DFA 放置到树的结点上,匹配概率大的离根部较近,概率小的离根部较远,调度时从根部开始进行.

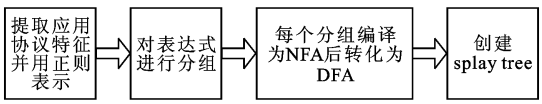


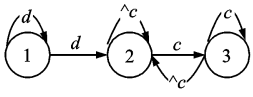
图 1 基本流程图

2.1 表达式集合分组

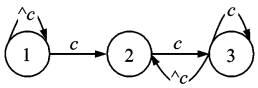
当将所有的表达式直接编译为一个组合 DFA 时,由于状态空间爆炸问题的存在,无法用硬件实现,而将每个表达式分别编译为一个 DFA 时,在一定程度上可以缓解状态空间爆炸问题,但对于每个输入字符则需要检查多个 DFA,又带来了计算复杂度较高的问题,影响了匹配速率.针对此问题,本文首先对表达式集合进行分组,在状态数量与计算复杂度之间选择一个较好的折中方案.另外,分组时如果将两个存在交叠的表达式放置于同一组中,则会引起 DFA 的状态数量急剧增长.

图 2a～2c 分别为表达式 r1:. $*d$. $*c$;r2:. $*e$. $*c$;r3:. $*a+(a|b)\{2\}c$ 所对应的 DFA,当把表达式 r1、r2 分为一组,r3 单独放置到一个组时,两个 DFA 所需的状态数量为 16,而将 r2、r3 分为一组,r1 单独一组时,两个 DFA 所需状态总数为 21.这是因为当数据流中出现字符 e 时,需要对状态复制两次.因此,为了避免类似情况的发生,本文在预处理阶段,提出一种高效的分组算法,隔离相互之间

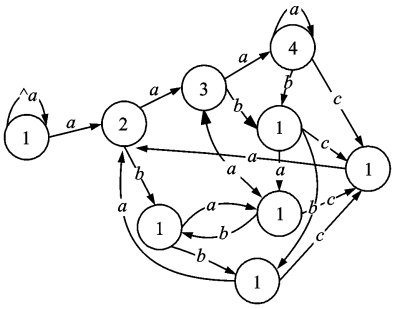
存在交叠的表达式，同时使处理每个字符的代价最低. 算法采取迭代对集合进行分组，每次迭代合并处理代价最低的两个组，通过迭代得到处理代价最小的相应分组.



(a)r1



(b)r2



(c)r3

图 2　表达式 r1、r2、r3 对应的 DFA

为了得到最优的分组，必须对处理每个字符的代价进行定量估计，以便得到处理代价最小的表达式而合并为一个组. 本文以单个 DFA 状态所对应的平均 NFA 状态数 n 作为估计代价标准，因此单字符处理代价可表示为下式

$$\mathrm{cost}(\{R_1,\cdots,R_k\}) = \sum_i n \tag{1}$$

在得到处理字符代价的前提下，可以采用如下分组算法，迭代完成后，可得到表达式集合的最低处理代价分组. 分组算法 GroupRSET (R)为：

(1)Input：R//正则表达式集合
(2)Output：groups with minimum processing cost.
(3)min_group＝cur_group＝{{r}：$r\in R$}；//min_group：最小代价组
(4)min_cost＝cost(min_group)；
(5)while the size of cur_group＞1　do//所包含组数大于 1
(6)mini_cost'＝＋∞；
(7)for each group pairR_1，R_2 in cur_group do
(8) cur _ group' ＝ cur _ group － {R_1，R_2} ∪ {$R_1 \cup R_2$}；
(9)/＊合并 group1、group2＊/
(10)　　if cost(cur_group')＜ min_cost' then
(11)　　min_group'＝cur_group'；
(12)　　min_cost'＝cost(cur_group')；
(13)　　end if
(14)end for
(15)if min_cost'＜ min_cost then
(16)min_group＝min_group'；min_cost＝min_cost'；
(17)end if
(18)cur_group＝min_group'；
(19)end while
(20)return min_group.

2.2　gDFA 的调度

分组完成后，对每个分组建立 DFA，即 gDFA，下一步就需要对 DFA 进行高效的调度. 传统的 DFA 多数以固定的顺序存储于链表当中，然后采用线性调度方法对 gDFA 进行调度，以实现对网络业务流的识别. 但是，这种调度方法效率低下，进行一次成功的匹配需要花费较长的时间，且吞吐量较低. 另外，由于正则表达式匹配本身就需要消耗较多的资源，故这种调度方法加剧了正则表达式匹配算法问题的严峻性. 为了提高正则表达式的匹配速率和吞吐量，本文采取将匹配概率大的分组 DFA 进行优先调度. 由于网络流是动态变化的，匹配概率也会随着网络流的变化而动态变化，相应的调度次序也会随着动态变化，因此本文做出以下假定：①DFA 是线性调度的；②当匹配到一个规则时就会输出此规则的地址并产生匹配信号，而不是继续对所有规则表达式进行匹配.

假定对规则集进行分组后产生 k 个分组，因此就有 k 个 gDFA，假设对于 gDFA$_i$($1\leqslant i\leqslant k$)在某个时刻的匹配概率为 p_i，匹配时间为 t_i，则平均匹配时间可表示为

$$T = p_1t_1 + p_2(t_1+t_2) + \cdots p_i(t_1+t_2+\cdots+t_i) + p_k(t_1+t_2+\cdots+t_i+\cdots+t_k) + (1-p_1-p_2-\cdots-p_k)(t_1+t_2+\cdots+t_i+\cdots+t_k) \tag{2}$$

可以证明：当满足 $p_1/t_1>\cdots>p_k/t_k$ 时，T 值最小；当所有 DFA 的匹配时间相等且大小均为 t 时，最小平均匹配时间为

$$T_{\min} = [p_1 + 2p_2 + \cdots + kp_k + k(1-p_1-\cdots-p_k)]t \tag{3}$$

此时，最优的调度顺序是 $p_1>p_2>\cdots>p_k$，即匹配

概率大的优先进行匹配.另外,为了适应网络流的动态变化,本文采取对分组 DFA 按概率大小构建 splay tree.这是一种动态二进制搜索树,它是由 Sleator 等[9]首先提出的,单次操作具有 $O(n)$的时间复杂度.其基本思想是:当一个结点被访问后,可通过一系列的旋转操作使它成为整个树的根;如果这个结点的位置很深,那么根到该结点的路径上也会有很多较深的结点.经过一系列旋转,这些结点的深度将会减小,从而起到了平衡整棵树的效果,且对于经常访问到的结点,它们的深度相对较小,访问它们的时间也较短.因此,利用伸展树的特性,可以达到自适应网络流变化的效果.

调度算法 ScheduleDFAs (TCP-links, Splay tree, gDFAs)为:

(1) compile each regular expressions groups into gDFA;
(2)construct the improved splay tree;
(3)insert the gDFAs into the splay tree;
(4)read the packets from the traffic;
(5) for all TCP-links output explicit application of search of TCP-links
(6) do
(7) traverse the splay tree from the root by level;
(8) if any DFA matching successfully, report the gDFA;
(9) stop the traverse; adjust the gDFA near the tree root;
(10) go back to the root of the splay tree;
(11) end if;
(12) end do;
(13) end for;
(14)end;

首先,将分组后创建的 gDFA 放置到树的各个结点,当某个 gDFA 产生匹配时,停止进行转移,将此 DFA 通过旋转操作移到离根结点较近的部位.算法流程如图 3 所示,通过一定时间的调整后,匹配概率大的 DFA 会位于根结点附近,每次匹配后所进行的操作次数会变得很少,能够较好地适应动态流的变化,匹配速率和吞吐量都得到了较大的提高.

3 实验仿真及分析

(1)实验环境:本算法采用 Intel(R) Pentium (R)4 CPU Duo 2.80 GHz,1.00 GB 内存,操作系统为 Windows XP,并采用 VMWare Workstation 6.5-7.0 搭建环境,规则来自于 Snort[10]提取的 150 条规则;实验数据流主要分为两部分,一部分由计算机产生,另外一部分来自于 NLANR[11].

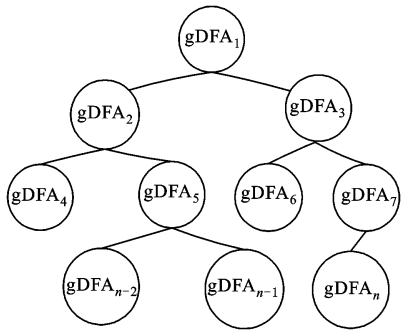


图 3 splay tree 调度 DFA 的流程图

在预处理阶段采用上述分组算法对规则分成 32 个组,构建成 32 个 gDFA 放置到 splay tree 中.本部分主要对分组前后状态数、吞吐量与 DFA 的平均调度次数等性能指标进行分析与对比(分组后构建的 gDFA 可采取压缩算法进行空间压缩,本文只讨论分组对 DFA 大小的影响).

(2)分组前后性能对比:实验中,首先将 150 条规则不进行任何分组构建为一个 DFA,计算其状态数、转移数以及空间消耗情况;然后,将此规则采用分组算法分为 32 个组,分别构造 32 个 DFA,进行状态数、转移数以及空间消耗的统计,与将所有的规则构造为一个 bigDFA 时的情况进行对比,如表 1 所示.

表 1 分组前后性能对比

自动机	状态数	转移数	空间消耗/kb
bigDFA	9 124	20 845	5 781
gDFA	6 932	15 739	4 237

由表 1 可以看出,通过分组算法分为 32 个组后,由于将存在交叠的表达式相互隔离了,故状态数减少了 1/3 左右.分组算法同时还考虑到了所需代价,故对转移数的减少也有较为明显的效果,存储空间消耗得到了明显的降低.

(3)预处理复杂度分析与对比:本部分实验通过设置每组状态机的最大代价大小,将 150 条规则分为不同的组数,得到其预处理的复杂度对比情况,如表 2 所示.可以看出,当设置的代价极限值较低时,分的组数较多,交叠表达式的隔离度也较大,所需状态数就比较少.分组数量越多,所需要的比较次数越

少，状态机中存在的交叉转移越少，其计算复杂度越低，编译时间也越短. 分组较少时，计算复杂度较高，编译时间就会变得比较长. 对于通常的网络设备来说，规则的更新并不是十分频繁.

表 2　分组复杂度对比

gDFA 最大代价	分组数	状态数	编译时间/s
285	32	6 932	18.7
575	16	7 563	47.2
1 200	8	8 214	125.3
2 300	4	8 993	650
9 150	1	9 124	893.6

(4)吞吐量大小分析与对比：本部分实验主要针对分组数对吞吐量的影响. 如图 4 所示，当初期阶段分组数为 1 时，则不进行分组，此时将所有表达式构建为一个 bigDFA，故存在交叠的表达式较多，从而造成吞吐量较低；随着分组数的增多，由于对存在交叠的表达式的相互隔离，吞吐量呈线性增长，当达到 32 个分组时，吞吐量达到最大值；随着分组数的进一步增多，所需的 DFA 数量也越来越多，这使得每个字符的处理代价不断提升，限制了吞吐量的进一步提高.

另外，本部分将 splay tree 调度方法与传统的线性链表调度算法进行了对比，其中线性调度时不采取分组算法，按单条规则构造单个 DFA 的方式构造多个 DFA，通过仿真得到线性调度的平均调度次数约为 45.3，而采用 splay tree 调度的平均调度次数约为 12.8，大大提高了调度效率.

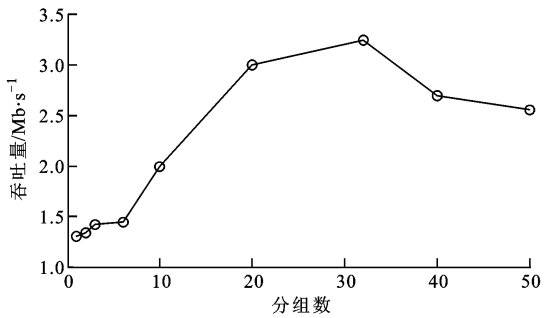


图 4　吞吐量随分组数的变化情况

图 5 为两种调度方法的吞吐量随规则数的变化情况对比. 实验仿真表明，与不进行分组时所采用的线性链表调度方法相比，采用分组构建 splay tree 进行调度能够自适应网络流的变化，吞吐量也提高了 3 倍左右. 另外，splay tree 调度方法比较适合规则数较大的场合，而线性调度方法比较适合规则数较小的情况，随着规则规模的扩大，线性调度方法吞吐量下降严重.

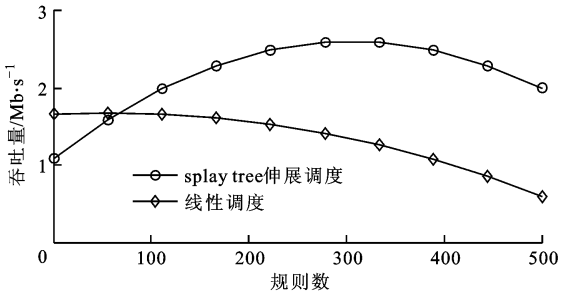


图 5　两种调度方法吞吐量随规则数变化的比较

4　结　论

本文针对网络流的特点和当前采用多正则表达式匹配进行网络业务识别时存在的效率低下、吞吐量低的问题，提出一种面向网络流的自适应多正则表达式匹配算法. 该算法采用一种高效的分组方法在 DFA 的大小和处理代价之间选择了较好的折中方案，隔离了存在交叠的表达式，减少了状态数量，控制了处理代价；然后，对每个分组构建 gDFA，按匹配概率大小创建 splay tree，能够根据网络流的变化动态调整结点顺序，匹配概率大的优先调度，大大提高了匹配效率，且吞吐量提高了 3 倍左右. 本文方法目前还不支持对网络中存在的压缩流的处理，这是下一步需要研究的工作.

参考文献：

[1] AHO A V, CORASICK M J. Efficient string matching: an aid to bibliographic search [C] // Proceedings of Communications of the ACM. New York, USA: ACM, 1975: 333-343.

[2] YU F, CHEN Z, DIAO Y, et al. Fast and memory-efficient regular expression matching for deep packet inspection [C] // Proceedings of ACM/IEEE Symposium on Architecture for Networking and Communications Systems. New York, USA: ACM, 2006: 93-102.

[3] BECCHI M, CADAMBI S. Memory-efficient regular expression search using state merging [C] // Proceedings of INFOCOM. Piscataway, NJ, USA: IEEE, 2007: 1064-1072.

[4] FICARA D, PIETRO A D, GIORDANO S.

Differential encoding of DFAs for fast regular expression matching[C]// Proceedings of IEEE/ACM Transactions on Networking. Piscataway, NJ, USA: IEEE, 2011: 683-695.

[5] KONG S, SMITH R, ESTAN C. Efficient signature matching with multiple alphabet compression tables[C]// Proceedings of Secure Comm. New York, USA: ACM, 2008: 1-10.

[6] JIANG Junchen, XU Yang, PAN Tian. Pattern-based DFA for memory-efficient and scalable multiple regular expression matching [C]// Proceedings of IEEE ICC. Piscataway, NJ, USA: IEEE, 2010: 1-5.

[7] XU Kefu, TAN Jianlong, GUO Li. Traffic-aware multiple regular expression matching algorithm for deep packet inspection [J]. Journal of Networks, 2011, 19(3): 799-806.

[8] FICARA D, ANTICHI G, PIETRO A, et al. Sampling techniques to accelerate pattern matching in network intrusion detection systems[C]// Proceedings of IEEE ICC. Piscataway, NJ, USA: IEEE, 2010: 1-5.

[9] SLEATOR D D, TARJAN R E. Self-adjusting binary search trees[J]. Journal of the ACM, 1985, 32(3): 652-686.

[10] Sourcefire. Snort network intrusion detection system [EB/OL]. [2011-10-06]. http://www.snort.org.

[11] NLANR. Abilene-I data set [EB/OL]. [2011-10-08]. http://pma.nlanr.net/Traces/long/ipls1.html.

（编辑 荆树蓉）