基于DoLFA的高效正则表达式匹配算法

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 郑州 450002)

摘 要 随着规则数量的急剧增长,表示正则表达式的DFA(Deterministic Finite Automata,确定型有限自动机)容易引起状态空间爆炸,难以满足高速网络的实时处理需求。提出一种高效的正则表达式匹配算法,该算法通过将正则表达式分割为精确串、字符集合以及重复字符3个子集,分别对其进行分区优化及检测,然后再利用结点信息对匹配信号进行连接,即构建一种特殊的状态机DoLFA(Divide-optimize-Link Finite Automata)。理论分析和仿真结果表明,该算法可以大大节省存储空间,并获得较高的吞吐量,且具有较强的扩展性。

关键词 深度包检测,正则表达式,有限自动机,编码,计数器
中图法分类号 TP393 **文献标识码** A

Efficient Regular Expression Matching Algorithm Based on DoLFA

DU Wen-chao CHEN Shu-qiao HU Yu-xiang
(National Digital Switching System Engineering & Technological R&D Center, Zhengzhou 450002, China)

Abstract With the rapid increase of the number of rules, the DFA used to present regular expression often results in states explosion, so it is very hard to satisfy the requirement of high speed network online processing. This paper proposed an efficient regular expression matching algorithm, which first divides an expression into three subsets: exact string, character class and character repetition, and then optimizes and detects the corresponding blocks, at last links them together with auxiliary node data structure, namely constructing a special state machine DoLFA. Theoretical analysis and simulation shows that this algorithm not only can save more memory space, but also provide high throughput performance and scalability.

Keywords Deep packet inspection, Regular expression, Finite automata, Coding, Counter

1 引言

深度包检测是网络安全与网络监管的关键技术,通常采用特征匹配算法,即将数据包内容与一组预定义的特征进行匹配。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AC[1]算法。随着检测内容的日渐复杂,具有强大的表达能力与灵活性的正则表达式逐渐代替精确字符串作为主要的报文特征描述方式。当前内容检测引擎已经开始支持大量的正则表达式,比如Snort入侵检测系统、Bro入侵检测系统等。

基于正则表达式的多模式匹配的原理是将正则表达式构造成有限状态机(FSM, Finite State Machine),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, Deterministic Finite Automata)和非确定性有限自动机(NFA, Nondeterministic Finite Automata),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。因此当前大部分研究都是基于如何压缩DFA的存储空间,如引入缺省路径的D^2FA[2]、基于状态合并的自动机[3]、混合型自动机即Hybrid-FA[4]等。但由于正则表达式由不同的正则特征组成,很难用单独一种检测引擎高效地对其进行匹配,因此算法存在检测效率低的缺点。

本文从网络安全检测的现状出发,提出了一种基于DoLFA分区处理的正则表达式匹配算法,即将正则表达式分割为精确串、字符集合和重复字符3个子集,构造一种特殊的状态机DoLFA,将这3个子集采用不同的检测引擎进行检测。该算法的最大优点是避免相互之间引起交叠,在大大减少内存需求的基础上提高了检测效率。本文第2节对相关工作和研究背景进行简单介绍;第3节对基于DoLFA的正则表达式匹配算法原理进行详细介绍;第4节是算法的性能分析;第5节给出实验结果;最后是结论和展望。

2 相关工作和背景

正则表达式匹配算法是计算机科学与技术领域中的经典问题。为适应高速网络数据包内容的检测,近年来,研究者从

到稿日期:2011-11-16 返修日期:2012-03-16 本文受国家重点基础研究发展计划(2012CB315901)和国家科技支撑计划(2011BAH19B01)资助。

杜文超(1984—),男,硕士,主要研究方向为深度包检测、模式匹配,E-mail:glossmen@163.com;陈庶樵(1973—),男,教授,主要研究方向为宽带信息网;胡宇翔(1982—),男,博士,主要研究方向为高速交换与调度。

状态和迁移边等方面提出了多种基于 DFA 的正则表达式匹配算法。在状态压缩方面，Becchi 等人[3]提出基于状态融合的 DFA，即采用迁移边标记方法来融合多个非等价状态，减少 DFA 的存储空间需求，且确保其最坏情况下的性能；Yu 等人[5]针对 Snort 规则提出两条改写规则，即把 DFA 状态数存在指数膨胀的正则表达式改写成线性增长的形式，并采用分组启发方式将一个规则集分割为多个子集并建立 Multi-DFA，但该算法适用性有限；Smith 等人[6]提出一种基于 XFA 的正则表达式匹配算法，即在状态上增加辅助变量来记录部分匹配结果，执行简单操作指令来检查匹配是否成功，避免了 DFA 状态空间爆炸问题；Kumar 等人[7]后来采用启发式方法消除正则表达式匹配算法的“失眠症”即不活跃状态占用许多片上存储空间、“健忘症”即需要大量的额外状态来记录部分匹配结果，以及“失算症”即 DFA 无法记录相同子串的匹配次数。

在迁移边压缩方面，Kumar 等人[2]提出带有默认前移的 DFA 表示法 D2FA，它通过合并状态之间重复的转移边来降低存储消耗，但只适合整齐二维表的情况，存在匹配吞吐量低和构建开销高的缺点；Becchi 等人[4]后来又提出基于混合型 HybridFA 的正则表达式匹配算法，即把每个正则表达式分为 head、tail 两部分，head 部分构建 head-DFA 作为预过滤，而容易引起状态空间爆炸的 tail 部分根据性能可构建 DFA 或 NFA。这种方法虽然缓解了存储空间消耗过高的问题，但仍存在着吞吐量低、处理速率不高的缺点；Masanori 等人[8,9]提出基于 LaFA 的正则表达式匹配算法，它将正则表达式不同的子集采用不同的检测引擎进行处理，消除了大量的冗余状态，并且可扩展性较强，但对存储空间的优化程度还不够，仍具有可改进之处。本文提出的基于 DoLFA 的正则表达式匹配算法就是在 LaFA 基础上进行了改进，它对各个子系统的优化程度更高，对存储空间的消耗更低。

3 基于 DoLFA 的正则表达式匹配

算法主要分为两部分：子集检测和子集连接。整体系统框图如图 1 所示，通过把正则表达式分割为精确串、字符集合以及重复字符 3 个子集，合并相同的元素，构造成一种特殊的状态机 DoLFA，如图 2 所示。子系统处理完毕后发送一匹配信号到 DoLFA，DoLFA 接收到信号后查找下一个需要处理的子集。精确串的匹配主要通过合并共同子串和按长度大小分区存储的方法进行优化；对字符集合采取按集合的包含与被包含关系构建二进制前缀树并进行编码；对于重复字符则引入计数器并附加转移条件，大大减少了状态数量。该算法的最大优点是对易被检测且出现频率较低的部分优先处理，避免频繁的访问字符集合和重复字符等较为复杂的检测系统；另外，可以合并所有表达式中大量相同的元素，对各种正则特征进行分区处理，以减少歧义路径和非歧义路径交叉产生的额外状态。

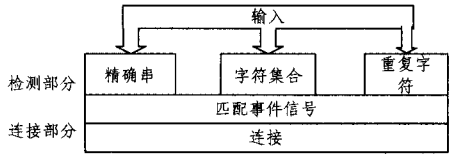

图 1 正则表达式整体系统图

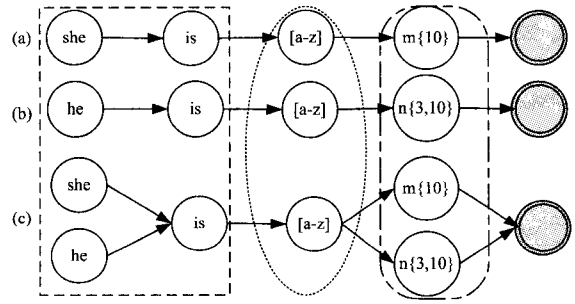

图 2 表达式基本分区

3.1 DoLFA 的构建

子集检测分为 3 个子系统进行检测：精确串检测(String Detection，SD)、字符集合(Character Class Detection，CCD)、重复字符(Character Repetition Detection，CRD)。首先按照精确串、字符集合、字符计数的顺序构建一种特殊的状态机 DoLFA，如图 2 所示。直角矩形内为精确串，椭圆内为字符集合，圆角矩形内为重复字符(计数结构)，图中灰色阴影部分表示可接受状态。如图 2(c)所示，可以合并相同的子串(is)和字符集合([$a-z$])，减少其重复检测次数。DoLFA 不同于传统的状态机，它采用 NFA 的构造方法，但每次可以接受多个字符，吞吐量较大。每个圆圈代表一个 large state，如图 2 所示，将{she，he，is}分别合并为 3 个 large state，可以减少所需状态数量。

3.1.1 SD BLOCK

精确串匹配是一个研究相对比较成熟的问题，具有很多较好的优化方法。DoLFA 是基于匹配事件的，当一个精确子串匹配成功后，会产生一个匹配事件。此匹配事件应包含以下信息：(1)子串的 ID 号；(2)该子串在数据包中出现的位置；(3)匹配信号能够持续的最短时间和最长时间；(4)状态信息以及额外信息。

精确串的处理具有很多优化方法，本文采用比较适合此算法的方法。如图 3 所示，两个正则表达式 R1、R2 中精确串{sheis，heis}含有共同部分 heis，而精确串{pattern，betters，latten}共同部分为 tte，而{pattern，betters}的共同部分为 tter。这两个共同子串存在重叠，因此不能同时被接受和合并。在此本文采用贪婪算法[15]从共同子串中选择符合条件的部分合并。给定一个规则集，假定可能存在 m 个不同长度的共同子串，其长度分别为 $l_1, l_2, \cdots, l_m$，对于第 i 个共同子串，假设有 t 个规则共同拥有，则定义 $l_i * t_i$ 为权值，此贪婪算法的基本步骤为：

(1)从表达式切割的精确串当中选择权值最大的共同子串，如果此权值为 0，不存在共同子串，返回；

(2)删除可能与(1)中选定的共同子串发生冲突或重叠的子串；

(3)已经选定的共同部分重新评估后返回步骤 1；

(4)共同子串选择完毕后进行合并。

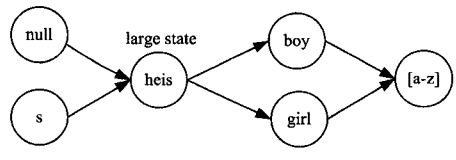

图 3 共同子串合并后的状态图

经过上述贪婪算法的处理后，{pattern，betters，latten}只能选定 tte 作为共同子串，此时权值为 9。得到精确串中含有的满足条件的共同子串后，对各个子串进行编号，分配一个状态号，然后合并精确串中的共同子串，合并后的精确串状态机如图 3 所示。存储时采用分区的存储方法，此时只需对共同子串存储一次，大大节省了存储空间。假定每次处理的最大子串长度为 5，对于长度小于 5 的子串按长度大小依次放入各自的分区当中，长度为 5 的子串存储于一个区当中，避免浪费存储空间。

3.1.2 CCD BLOCK

字符集合的处理是在普通的二进制树的基础上，首先提取正则表达式中的字符集合，然后遵循两个基本原则：(1)首先将无重叠的字符集合置于树结点；(2)按照集合范围大小的顺序放置，范围较大的优先放置，离根部较近。比如：字符集合[$a-z$ $A-Z$]>[$a-z$]或[$A-Z$]，因此[$a-z$ $A-Z$]更靠近于树的根结点；如果字符集合 A 包含字符集合 B，则 B 必定为 A 的后代。

图 4 为从正则表达式中提取的字符集合构造的二进制树。[0—9]与任何字符集合都不重叠，所以优先放置到树中；而[$A-z$ $a-z$]所包含的范围最大，[$A-z$]和[$a-z$]均为它的最大子集，分别放置到它的左右孩子结点；[$B-H$]和[$T-Z$]均为[$A-Z$]的子集，也分别放置到其孩子结点，按照左结点为 1、右结点为 0 的原则编码，所形成的二进制树即为图 4，字符集合所对应的编码如表 1 所列。但是有些字符集合可能具有相同的编码，为了区分它们，在编码后面增加其结点深度变量 *depth*；[$B-H$]对应的编码为 111/3；[$A-Z$ $a-z$]对应的编码为 1/1。

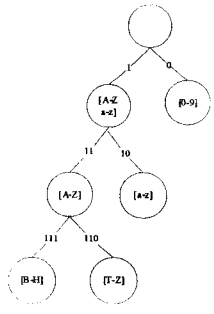

图 4 字符集合构造树与编码

表 1 字符集合所对应编码

范围	编码 tag	长度
[A—Z a—z]	1＊＊	1
[0—9]	0＊＊	1
[A—Z]	11＊	2
[a—z]	10＊	2
[B—H]	111	3
[T—Z]	110	3

字符集合编码可以分为 4 个基本步骤：(1)从正则表达式中提取模式串，保证每个模式串中至少含有一个字符集合；(2)从模式串中提取字符集合；(3)根据所提取的字符集合构建前缀树；(4)创建相应的数据结构(编码及码长)。但在步骤(3)中，有时所提取的字符集合不一定满足构造二进制树的条件，此时需要借助于增加虚结点，如图 5 所示，白色结点为虚结点。

二进制前缀树的构造流程图如图 6 所示，可采取迭代法构造，通过引入 3 个变量 tem1、tem2、tem3 来存放无重叠字符集合，并对 3 个变量所表示的字符集合进行并运算得到一个累计范围 R。在迭代时，按照前面所述的原则，优先放置范围较小的字符集合。在每次迭代结束后，如果累计范围等于 tem1，将 tem1 放置到树上；如果累计范围包含 tem1，将 tem1 和累计范围一起放置到树中，累计范围置于母结点，tem1 置于相应子结点；如果每次迭代结束后，tem2 中有变量，而 tem3 是空的，则也将 tem2 置入树中。如图 6 所示，在第一次迭代中，首先将[$A-E$]存于 tem1 中，此时累计范围等于 tem1，然后选取[$F-K$]放入 tem2 中，并对累计范围进行更新。第一次迭代结束后，各个变量的值如图 6 Step 4 所示。然后按照构造树的几个原则，此时累计范围包含 tem1，将 tem1 放置到子结点，累计范围放置到母结点，如果累计范围不存在，此时母结点为虚结点，最终迭代后得到如图 5 所示的二进制树。

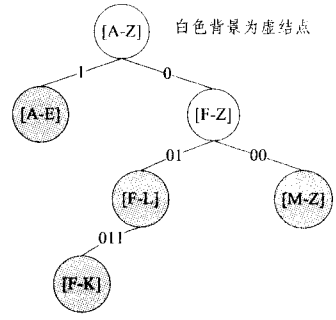

图 5 带有虚结点的二进制字符集合树

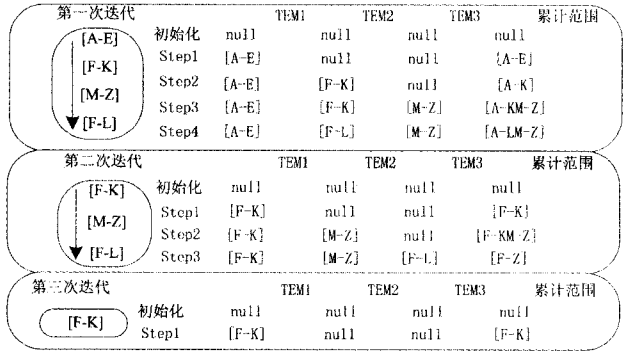

图 6 二进制前缀树的迭代构造流程

字符集合处理系统流程如图 7 所示，采取 3 个表进行操作：编码表、长度表、查表结果列表。首先根据输入字符查找编码列表中对应的编码，编码长度表中包含所有输入字符所在字符集合对应的编码长度组合，对编码作 HASH 寻址，HASH 函数可从文献[16]选取；查找到所在字符集合对应的编码长度组合，然后再将编码和码长一起作 HASH 寻址得到查表结果。由于 HASH 存在不可避免的冲突，可以设置多个哈希函数来避免冲突。

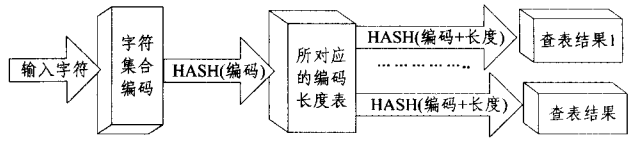

图 7 字符集合处理流程

此部分通过构造前缀树进行编码，大大降低了字符集合检测系统的复杂度，存储时也只需存储集合编码以及码长，与传统的状态机所需大量状态相比，大大节省了存储空间；另外

此种处理方法的处理速率也较快，规则的更新也较为方便。

3.1.3 CPD BLOCK

含有重复字符的正则表达式如果用传统的 NFA 表示，所需状态与 n（n 为表达式所包含字符数）值成线性关系，当 n 值很大时，所需状态数量也会变得很大；采用 NFA 的最大问题是并行处理时有多个状态同时被激活，对内存带宽要求较高，并且处理速率较慢。而采用普通 DFA 则需要把 NFA 通过子集构造法转换为 DFA。由于 DFA 的 n 个状态可能与 NFA 的 2^n 个状态相关，通过子集构造法将 NFA 转换为 DFA 时会引起状态空间爆炸，特别是当表达式前缀中同时含有字符集合和计数结构时，状态数量呈指数增长，这对算法的实现是很大的挑战。

针对上述问题，此部分引入计数器对重复字符进行处理，且与用普通的状态机功能完全等价。以子表达式. $*a.\{n\}bc$ 为例，前面的. $*$ 表示可以出现在输入字符的任何位置，也就是需要检测包中任何位置是否包含字符 a 后面为 n 个任意字符且这 n 个任意字符后面是 bc 的字符串。前面的. $*$ 用一个 large state 表示，如图 8 中的状态 0、1、3，而状态 2 被称为 counting state，此时状态的转移需根据计数器的值和输入字符决定。为了避免单一的计数器容易出现漏报的情况，此时采取两个线程计数。例如假定表达式中 $n=3$，若要对数据内容 *amanbobc* 进行检测，当只设置一个计数线程时，字符 a 能够匹配，后续是 3 个任意字符，然后是字符 b，但在字符 o 处失配，此时会产生一个失配信号，然后放弃处理这段数据，继续向前处理后续数据，从第二个字符 a 开始时可以实现完全匹配。因此为了避免漏报情况，可以在第一个 a 和第二个 a 出现时分别设置一个计数线程。如图 8 为计数型 NFA 结构，虚线转移 1-2 代表计数开始（$cnt=0$），当达到 n 时停止计数，即状态转移为 2-3，其中（|）后面代表转移条件，当输入为 *amanbobc*，状态转移表可表示为表 2：第几个计数线程在处理字符 n 的时候开始被激活进行计数，当字符 b 到达时，由于 $cnt1=3$，成功从状态 2 进入状态 3，但此时并不能实现完全匹配，此时 $cnt2$ 并不等于 3，将转移引向 2-2，最后实现完全匹配。其中 $cnt=+$ 表示一个线程计数结束，还有另一个计数线程处于激活状态。

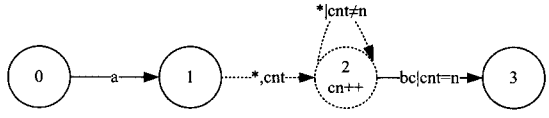

图 8 表达式. $*a.\{n\}bc$ 对应的计数器型 NFA

表 2 计数型 NFA 基本状态转移表

输入字符	激活状态	计数线程	转移条件
a	0,1		
m	0,2	cnt1=1	cnt≠n
a	0,1,2	cnt1=2	cntn
n	0,2	cnt1=3 cnt2=1	cnt=+
b	0,2,3	cnt2=2	cnt≠n
o	0,2	cnt2=3	cnt=n
b	0,3		

如图 8 所示，此时只用一个 counting state 代替 n 个单独状态，可以大大减少状态数量，通过子集构造法可以将计数型 NFA 转化为计数型 DFA，消除了状态空间爆炸。

3.2 基本流程

当数据流进入检测系统后，如果能够匹配，则会发送一个匹配信号，此时匹配信号即为该子集的 ID（SID），DoLFA 接收到此信号后，找到正在处理的结点。通常的正则表达式集往往包含成千上万条规则，规则进行切割优化后，采取结点进行管理，结点的基本数据结构为：(1)子串的 ID 号(SID)；(2)子串所在的表达式规则 ID 即 RULEID；(3)该规则有效检测的相对最长时间和最短时间；(4)指向下一跳状态的指针。另外此部分还需设置一个时间存储器，用于存储子集被检测到的绝对时间，当一个子集被检测到时，我们将用时间存储器获取的绝对时间与相对时间进行比较，看是否为有效检测。如当输入为 sheisxyzgirl 时，可对其进行时间信息编号，如表 3 所列。

表 3 sheisxyzgirl 时间信息编号

Input	s	h	e	i	s	x	y	z	g	i	r	l
Time	1	2	3	4	5	6	7	8	9	10	11	12

以表达式 R1：. $*$ sheis[$a-z$]{3}girl 和 R2：. $*$ heis[$a-z$]{3}boy（. $*$ 代表可以出现在包内容的任何位置）为例，首先表达式被切割为 s1：s；s2：heis；s3：girl；s4：boy；以及 v1：[$a-z$]，其中 s1 存储于字符长度为 1 的区之中，s2、s3 存储于字符长度为 4 的分区之中，s4 则存储于长度为 3 的分区之中，进行切割与精确串的合并优化后可以构建如图 3 所示的状态机。为了构建高效的状态机，引入 null state 和 large state，null state 常位于首状态，large state 为合并字符后的状态。此时以 0 或 1 代表非激活与激活信号，首先精确串 s1 对应的部分被激活，在时间 1，s1 与输入内容产生匹配，记录其匹配时间，产生一匹配信号即 SID 传送到状态机，状态机接收到信号后，查到该正则处理此字符串的结点，然后根据其指针找到下一个需要处理的子集 s2：heis，其相应的处理系统被激活，如果匹配，记录其匹配时间，并根据时间距离验证是否为有效匹配，如果为有效匹配，则激活字符集合处理模块，字符集合则根据构造树编码进行匹配；如果不是，则输出失配信号，继续往前处理后续输入字符。处理完成后根据结点中 RULEID、时间信息证实是否能够构成一个完全匹配信号。为了避免两个意义相近的表达式相互影响，可以采用并行化处理将其置于不同的处理线程当中，以免同时操作降低检测速率。

4 算法性能分析

本算法的性能分析主要针对空间消耗和吞吐量两个主要指标，然后将本算法的性能与现有算法进行比较。本算法主要针对正则表达式，与输入字符串无关，因此在最坏条件下仍具有稳定的性能。

4.1 内存空间消耗分析

由于本算法分 3 个子系统进行处理，其状态和结点信息分别存储于各自的存储器当中，对其内存消耗进行讨论。

4.1.1 SD BLOCK 空间消耗

由于此处引用贪婪查找算法进行共同子串查找与合并，可以对共同子串进行合并，假设可以合并的长度为 l_i（$l_i \geqslant 2$），有 t_i 个精确子集共同拥有，则可以节省 $(t_i-1)*l_i$ 个字符空间和 $(t_i*l_i)-1$ 个状态。假定此处如果用普通 DFA 表示所需状态数量为 m，所需转移数量为 n，则普通 DFA 的存储空间消耗可表示为：

$$M_{1DFA}=n(\log_2 m/8+32)+m \quad (1)$$

假定用 DoLFA 表示可以节省的状态数为 m'，可节省的

转移数量为 n'，则此部分用 Do-LFA 的存储空间消耗可表示为：

$$M_{1DoLFA}=(n-n')(\log_2(m-m')/8+33) \quad (2)$$

4.1.2 CCD BLOCK 空间消耗

此部分主要通过 3 个表：字符集合编码表、码长列表、查表结果表来完成，编码表的大小可通过 $l*256$（l 为编码长度）来计算；而码长表主要存储合并后的字符集合编码信息和长度信息。码长列表和查表结果列表仅存储被激活的正则表达式对应的长度信息，所需空间很小，假定为 s；同时假定通过正则表达式可提取的字符集合数为 x，而经过二进制树编码后需要存储的有效的字符集合数为 x'，则可节省的存储空间为：

$$M_{csave}=(x-x')*l*256-s\text{bits} \quad (3)$$

4.1.3 CPD BLOCK 空间消耗

由于本部分采用辅助计数器的方式解决重复字符问题，正则表达式中每个计数单元只需一个状态（counting state），若需重复次数为 n，与普通 DFA 相比，则每个计数单元可节省 $n-1$ 个状态，但考虑到需引入辅助转移条件，假定此部分单个辅助变量所需空间为 s 字节，表达式长度（表达式所包含字符数）为 l，每个状态所需字节数 w，则单个计数单元可节省的存储空间为：

$$M_{csave}=(n-1)*w-s*l \text{ 字节} \quad (4)$$

连接部分由于采用结点进行操作，此时结点大小一般为 96bits，假设所需结点数为 k，则结点存储器空间消耗为：$96*k$ bits。

4.2 搜索性能分析

由于 DoLFA 分 3 个子系统进行处理，精确串的处理每次可以处理多个字符（假定为 w 字节），需要送入精确串部分处理的子集的总字符数为 m，则所需转移数近似为：

$$n=\lceil m/w \rceil \quad (5)$$

同时字符集合通过编码和 HASH 寻址，重复字符通过设置计数器，与普通状态机单字符一次转移相比，其所需转移数大大减少，吞吐量和处理速率也得到大大提高。另外此系统适合于并行处理，对于搜索速率的提高有很大的空间。

5 实验仿真

仿真实验规则库来自于 Snort[16] 的 ftp. rules、http. rules、imap. rules、smtp. rules、spyware. rules 5 个规则集，每个规则集含有 300 条规则，共 1500 个正则表达式。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[17]，通过构建 DoLFA 和进行实验仿真可得到这 5 个规则集的空间消耗情况，如表 4 所列。与普通的 DFA 相比，DoLFA 通过对表达式的切割和分区优化处理，大大降低了空间消耗，有效地消除了元字符交叠引起的空间膨胀。

表 4 DoLFA 空间消耗

RegEx	ftp. rules	http. rules	imap. rules	smtp. rules	spyware. rules
Size	300	300	300	300	300
DoLFA	487kbits	495kbits	421kbits	576kbits	472kbits
DFA	11790kbits	37836kbits	10532kbits	15379kbits	22934kbits

在不同算法的空间消耗对比实验中，选取与 Multi-DFA、Hybrid-FA 和 LaFA 3 个状态机进行对比，其中 Hybrid-FA 算法的 DFA 头部结构深度设置为 10 字节，Multi-DFA 算法采用 4 个状态机实现，由此得到 4 种算法的空间消耗对比，如图 9 所示。其中 Hybrid-FA 采用 DFA 和 NFA 的混合结构，以部分搜索带宽为代价压缩状态转移图；Multi-DFA 采用分组的方式，最大限度地减少因规则交叠而引起的空间膨胀，当规则规模较大时，分组的作用将逐步降低；LaFA 虽然通过切割表达式大幅降低了空间消耗，但对于复杂表达式的处理效率较低。而 DoLFA 在 LaFA 基础上，通过对共同子串的合并、字符集合编码和引入辅助计数器的优化，使其压缩云效果较为明显，特别是规则数量较多时。

在搜索性能的仿真试验中，实验数据源中包含检测特征的数据包称为攻击包，攻击包数量与测试数据包总数的比值称为攻击强度。由于 4 种算法搜索方式的差异，综合考虑搜索带宽和状态机数量等因素，实验采用单包的平均转移次数衡量算法的搜索能力，不同攻击强度下单包平均转移次数对比如图 10 所示，Multi-DFA 与 Hybrid-FA 设置基本不变，单包转移次数较大。与其他算法相比，DoLFA 明显降低了转移次数，搜索性能有较为明显的提升。

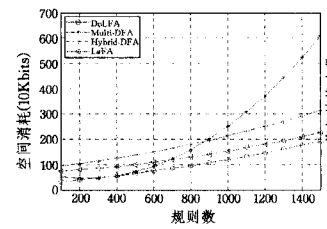

图 9 四种算法的空间消耗对比

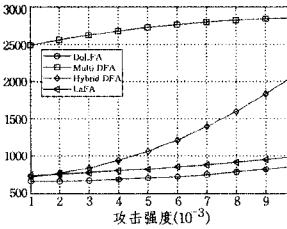

图 10 不同攻击强度下的平均转移次数

结束语 当前正则表达式匹配算法随着规则规模的增大，状态空间急剧膨胀，对存储资源的消耗过大，以及传统的状态机都是基于单字符和单状态，造成吞吐量有限。本文打破传统的状态机结构，提出一种将正则表达式分割为精确串、字符集合和重复字符 3 个子集，构建特殊状态机 DoLFA 的方法，分别对这 3 个部分进行优化处理，避免相互之间引起交叠，大大节省了存储空间，同时将容易被检测的精确串优先处理，以过滤掉大量不可能的匹配，减少对重复字符等一些复杂正则特征的操作次数。与现有的正则表达式匹配算法相比，该算法不仅大大降低了存储空间消耗，能够容纳更多的规则，而且吞吐量更高，可扩展性也较强。

参 考 文 献

[1] Aho A V, Corasick M J. Efficient String Matching: An Aid to Bibliographic Search[J]. Communications of the ACM, 1975, 18(6): 333-340

[2] Kumar S, Dharmapurikar S, Yu Fang, et al. Algorithms to accelerate multiple regular expressions matching for deep packet inspection[A]// Proc. of SIGCOMM[C]. Pisa, IT: ACM Press, 2006: 339-350

[3] Becchi M, Cadambi S. Memory-efficient regular expression search using state merging[A]// Proc. of INFOCOM[C]. Anchorage, USA: IEEE Press, 2007: 1064-1072

[4] Becchi M, Crowley P. A hybrid finite automaton for practical deep packet inspection[A]// Proc. of the 2007 ACM CoNEXT Conference[C]. New York, USA: ACM Press, 2007: 1-12

[5] Yu F, Chen Z, Diao Y, et al. Fast and memory-efficient regular expression matching for deep packet inspection[M]. ANCS, 2006: 93-102

[6] Smith R, Estan C, Jha S, et al. Deflating the big bang: fast and

scalable deep packet inspection with extended finite automata [A]// Proc. of SIGCOMM[C]. Seattle, USA: ACM Press, 2008: 207-218

[7] Kumar S, Chandrasekaran B, Turner J, et al. Curing regular expressions matching algorithms from insomnia, amnesia and acalculia[A]// Proc. of ANCS[C]. Princeton, USA: ACM Press, 2007: 155-164

[8] Bando M, Artan N S, Chao H J, et al. LaFA: Lookahead Finite Automata for Scalable Regular Expression Detection [A] // Proc. of ANCS[C]. Princeton, USA: ACM Press, 2009: 40-49

[9] Bando M, Artan N S, Mehta N, et al. Hardware implementation for scalable lookahead regular expression detection[M]. RAW, 2010

[10] Cormen T H, Leiserson C E, Rivest R L, et al. Stein, Introduction to Algorithms (Second Edition)[M]. The MIT Press, 2002

[11] Carter J L, Wegman M N. Universal classes of hash functions (extended abstract)[M]. STOC, 1977: 106-112

[12] Becchi M, Crowley P. Extending finite automata to efficiently match Perl-compatible regular expressions [A] // Proc. of the 2008 ACM CoNEXT Conference[C]. Madrid, ES: ACM Press, 2008: 73-81

[13] 黄昆，张大方，谢高岗，等. 一种面向深度数据包检测的紧凑型正则表达式匹配算法[J]. 中国科学：信息科学，2010，40(2)：356-370

[14] 于强，霍红卫，等. 一组提高存储效率的深度包检测算法[J]. 软件学报，2011，22(1)：149-163

[15] Snort network intrusion detection system[EB/OL]. http://www.snort.org

[16] NLANR. Inder of/Traces/Traces/daily/20080801[EB/OL]. http://pma.nlanr.net/Traces/Tra ces/daily/20080801，2008-12

（上接第 4 页）

和总能耗。首先，在几种存储设备中，闪存的能耗的确是最低的。其次，内存的能耗基本与其容量成正比，所以当两类闪存缓存结构采用更小的内存时，内存能耗部分下降得非常明显。第三，除了混合存储层由于用低能耗的闪存替换了部分磁盘导致磁盘能耗显著降低外，其余情况的磁盘能耗相差不大，闪存缓存结构的磁盘能耗比传统结构略低一点，这是因为减少了一部分磁盘访问操作，但是由于磁盘的活跃状态和空闲状态的功率相差不大，因此磁盘节能的效果并不显著。最后，基于闪存的混合存储系统的确明显降低了总能耗，尤其是采用较小的内存时。

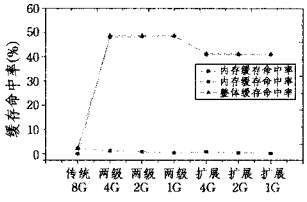

图 7 节能测试—缓存命中率

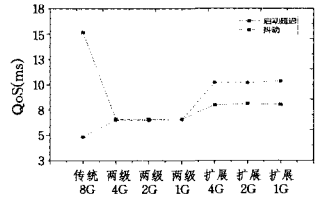

图 8 节能测试—QoS

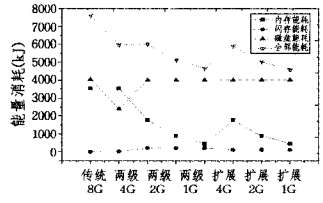

图 9 节能测试—能耗

综合以上分析，混合型流媒体存储系统引入闪存、减少内存的方式在提升性能和降低能耗方面的效果是非常显著的，是一种具有很高实用性并值得推广的方案。在多种混合型存储体系结构中，混合缓存的体系结构无论是在性能、成本，还是节能方面都要优于混合存储层结构；混合缓存中，两级缓存的性能略好，但扩展缓存对闪存设备的磨损较少，实际使用寿命更长，而二者在节能方面的表现非常接近。

结束语 本文设计并实现了一种基于闪存的混合型流媒体存储系统的仿真工具，其可以对多种典型的混合型存储系统进行准确、高效的仿真，并全面采集性能、服务质量、能耗等多方面的详细结果。通过性能测试和节能测试证明，混合型存储系统能显著提升系统性能，降低能耗。

参 考 文 献

[1] 中国互联网络信息中心. 中国互联网发展状况统计报告[EB/OL]. http://www.cnnic.net.cn/dtygg/dtgg/201107/W020110719521725234632.pdf. 2011-7

[2] Yarow J. Videos on Youtube grew 123% year over year, while Facebook grew 239% [EB/OL]. http://www.strangelove.com/blog/2010/06/videos-on-youtube-grew-123-year-over-year-while-facebook-grew-239, 2010-06

[3] Gal E, Toledo S. Algorithms and data structures for flash memories [J]. ACM Computing Surveys, 2005, 37(2): 138-163

[4] Ganger G, Ganger G, Worthington B, et al. The DiskSim simulation environment(v4.0) [EB/OL]. http://www.pdl.cmu.edu/DiskSim, 2009-09

[5] Agrawal N, Prabhakaran V, Wobber T, et al. Design tradeoffs for SSD performance[C]// Proc. USENIX 2008 Annual Technical Conf., 2008. Berkeley, CA, USA: USENIX Association, 2008: 57-70

[6] Kim Y, Tauras B, Gupta A, et al. FlashSim: A Simulator for NAND Flash-based Solid-State Drives[C]// Proc. First Int'l Conf. on Advances in System Simulation, 2009. Porto: IEEE, 2009: 125-131

[7] Bitar R. Deploying hybrid storage pools with Sun flash technology and the Solaris ZFS file system[EB/OL]. http://wikis.sun.com/download/attachments/190326221/820-5881.pdf, 2011-10

[8] Kgil T, Roberts D, Mudge T. Improving NAND Flash Based Disk Cache[C]// Proc. of the 35th Annual Int'l Symp. on Computer Architecture(ISCA) 2008. Beijing, China, 2008: 327-338

[9] Matthews J, Trika S, Hensgen D, et al. Intel® Turbo Memory: Nonvolatile Disk Caches in the Storage Hierarchy of Mainstream Computer Systems [J]. ACM Trans. Storage, 2008, 4(2): 24

[10] Pinheiro E, Bianchini R. Energy Conservation Techniques for Disk Array-Based Servers[C]// Proc. 18th Int'l Conf. Supercomputing. New York, USA: ACM, 2004: 68-78

[11] Intel. Intel® X25-E Extreme SATA Solid-State Drive[EB/OL]. http://www.intel.com/design/flash/nand/extreme/index.htm. 2011-11

[12] Xiao X, Shi Y, Zhang Q, et al. Toward Systematical Data Scheduling for Layered Streaming in Peer-to-Peer Networks: Can We Go Farther? [J]. IEEE Trans. Parallel and Distributed Systems, 2010, 21(5): 685-697

[13] 汪小梅，朱华. 一种改进的小波变换阈值去噪法[J]. 重庆理工大学学报：自然科学版，2010，24(6)：48-51