◎热点与综述◎

高性能正则表达式匹配算法综述

付 哲[1,2],李 军[2]
FU Zhe[1,2], LI Jun[2]

1.清华大学 自动化系,北京 100084
2.清华大学 信息技术研究院,北京 100084
1.Department of Automation, Tsinghua University, Beijing 100084, China
2.Research Institute of Information Technology, Tsinghua University, Beijing 100084, China

FU Zhe, LI Jun. Survey on high performance regular expression matching algorithms. Computer Engineering and Applications, 2018, 54(20):1-13.

Abstract: Deep inspection is playing an important role in providing secure network environment and guaranteeing network service quality. As a key technology for high-performance deep inspection, regular expression matching algorithms attract a great deal of attention in both academic research and industrial practice. With the rapid development of network technology, the volume of network traffic keeps increasing, the rulesets used for deep inspection are becoming larger while more complex, and the network topology is more flexible and dynamic than ever. Existing matching methods are facing many challenges, including those in matching speed, memory consumption, update ability, and so on. This survey first introduces the background of regular expression matching problem, then categorizes existing algorithms in the aspects of memory compression, speed acceleration, new automata design, and rule partition and grouping, and evaluates them in matching speed, memory consumption, as well as preprocessing with real life network traffic and rulesets. A guideline for efficient regular expression matching under different scenarios, and an outline of future work on high performance regular expression matching algorithms are also provided.

Key words: regular expression matching; finite automata; algorithm; evaluation

摘 要:深度检测在维护网络安全、保证服务质量等方面扮演着重要的角色。正则表达式匹配算法作为高性能深度检测的核心技术,具有重要的研究价值和实践意义。随着网络流量不断增长、规则数目持续增多以及网络结构日趋灵活和动态,现有的正则表达式匹配算法面临着匹配速度、内存占用和更新能力等多方面的挑战。介绍了正则表达式匹配算法的研究背景,从空间压缩、匹配加速、新型自动机设计以及规则拆分和分组四个角度入手,分类总结了学术界具有影响力的研究成果。通过基于真实网络流量的评测,比较了几种经典匹配算法在不同规则集上的匹配速度、内存占用和预处理时间等性能指标,并给出了不同需求场景下高效正则表达式匹配算法的选择建议,归纳了高性能正则表达式匹配算法的下一步发展方向。

关键词:正则表达式匹配;有穷自动机;算法;评测
文献标志码:A **中图分类号:**TP393 doi:10.3778/j.issn.1002-8331.1808-0339

1 引言

随着网络技术的飞速发展,互联网、云计算、移动通信、物联网等已深入到生产和生活中的各个方面,成为像水电气一样必不可少的基础设施,全球互联网用户已经突破了40亿[1],每月的互联网流量已经达到了121 694 PB[2]。伴随网络技术迅速发展和普及的是层出不穷的安全事

基金项目:国家重点研发计划(No.2016YFB1000102)。
作者简介:付哲(1991—),男,博士,主要研究领域为高性能算法,E-mail:zhefu0@outlook.com;李军(1962—),男,博士,研究员,主要研究领域为网络体系结构和网络安全。
收稿日期:2018-08-20 **修回日期:**2018-09-17 **文章编号:**1002-8331(2018)20-0001-13

件。近几年爆发的Heartbleed、WannaCry、Memcache放大攻击等安全威胁已经给互联网服务提供商带来数百亿的损失[3-4]。网络安全问题已成为社会无法回避、并且需要高度重视的关键问题之一。另一方面,网络应用的多样化和差异化,需要网络服务提供商能够提供满足不同服务质量的网络服务,云际网络更是为网络服务的发展提供了宽广的平台[5]。近年来,大量新型应用层出不迭,如基于点对点技术的视频分发、在线直播等,需要更加复杂和灵活的规则对其进行准确识别和管理。

网络流量的急剧增大和应用类型的迅速增多极大地提高了对网络流量进行检测和管理的难度。防火墙(Firewall)、入侵检测系统(Intrusion Detection System,IDS)、入侵防御系统(Intrusion Protection System,IPS)等是目前广泛使用的网络流量检测与管理设备。传统的防火墙需要对网包(packet)的第三层、第四层包头(header)信息进行检测,包括源IP地址、目的IP地址、源端口号、目的端口号、协议等五元组信息。然而,越来越多的特征隐藏在网包载荷(payload)中,仅使用包头五元组信息无法对流量进行细粒度识别,不足以检测出复杂的安全威胁。深度检测(Deep Inspection)应运而生,其不仅对网包包头进行检测,还会对网包载荷进行匹配。因此,深度检测技术广泛应用于流量检测和管理设备。一般说来,网络流量进入防火墙和IDS/IPS等设备中后,首先解析网包的五元组信息,分类为不同的网流(flow),根据管理策略表,执行不同的处理动作。对于需要进行深度检测的网流,相应设备会对网包载荷进行规则匹配,并根据匹配结果执行相应的操作。

早期,深度检测通常使用字符串匹配算法。经典的字符串匹配算法有Knuth-Morris-Pratt(KMP)[6],Boyer-Moore(BM)[7],Aho-Corasick(AC)[8]和Wu-Manber(WM)[9]等。随着网络服务和应用的不断发展,待检测的特征变得越来越复杂,难以用精确字符串进行准确的描述。正则表达式使用单个字符串可以描述一系列满足某个句法规则的字符串集合,因此其语义表达能力和灵活性远远高于精确字符串,逐渐成为深度检测中规则描述和匹配的首选方法,广泛应用于安全检测、应用分类、协议识别等领域。

高性能正则表达式匹配算法作为网络流量深度检测的关键技术,一直受到学术界和工业界的广泛关注。本文对近年来基于通用平台的高性能正则表达式匹配算法进行了全面的分析和总结,主要贡献包括:

(1)介绍了正则表达式匹配算法的研究背景,并总结了现阶段匹配算法面临的挑战。

(2)归纳了通用平台上高性能正则表达式匹配算法研究:从空间压缩、性能加速、新型自动机以及规则拆分和分组四个不同出发点总结了现有的研究成果,并分析比较各算法的优缺点。

(3)测试了高性能正则表达式匹配算法性能:以匹配速度、内存占用和预处理时间作为评测指标,通过真实网络流量比较了几种经典的匹配算法在不同规则集上的实际性能。

(4)提供了正则表达式算法的选用建议:针对不同的应用场景和需求,本文给出了高效正则表达式匹配算法选择方案。

2 背景与挑战

网络流量深度检测中,正则表达式由于其灵活、强大的表达能力,成为定义复杂特征的首选形式。然而,正则表达式匹配在多个指标上都面临着性能挑战,成为了深度检测的瓶颈。

2.1 正则表达式

从历史上来看,正则表达式最初是作为一个简单计算模型被理论计算机科学家于20世纪50年代提出的[10]。1968年,Thompson编写的文本编辑器QED首先实现了程序意义上的正则表达式匹配[11]。他和Ritchie随后创造的Unix,将正则表达式匹配引入了计算机的主流应用[12]。到了20世纪70年代晚期,正则表达式匹配已成为Unix的关键功能,成为ed、sed、grep、egrep、awk等工具中的核心方法。

一般而言,正则表达式由一系列ASCII(American Standard Code for Information Interchange,美国信息交换标准代码)字符构成,其中一部分作为元字符(Meta-character)。点号(.)、星号(*)和垂直符号(|)是正则表达式中应用最广泛的元字符。与普通ASCII字符不同,元字符表示特殊的含义。表1列举了正则表达式中最常见的若干种元字符以及它们的含义及示例。元字符给予了正则表达式丰富的表达能力:一条正则表达式可以匹配一组满足要求的精确字符串,而不仅仅是单条精确字符串。例如,Snort[13]利用正则表达式"[?&](search|topic)=[^&]*?(\x27|\%27)(\s*|(\%20)*)(\x3b|\%3b)"(见图1)检测Twiki[14]搜索函数的非法远程代码调用。这一特征无法用单条精确字符串描述。

2.2 自动机

在网络流量检测中,精确字符串匹配一般通过BM、WM、AC等匹配算法实现,或采用哈希、bloom-filter等方法进行加速。然而,传统的精确字符串匹配方法难以实现正则表达式中元字符所描述的语义,因此,正则表达式通常首先编译为有穷自动机(Finite Automata)进行匹配。理论证明,正则表达式和有穷自动机在数学上是完全等价的[15]。对于任何一个正则表达式r,总存在一个有穷自动机$\mathcal{A}$,其表达的语义与r完全相同;反

表1　正则表达式中常见的元字符含义及示例

元字符	含义及举例
.	任意字符 例如:“a.b”匹配“aab”,“abb”,“acb”等
[]	括号中的任意字符 例如:“[ab]”匹配字符“a”或者“b”
[^]	不在括号中的任意字符 例如:“[^ab]”匹配除“a”,“b”之外的任意字符
*	前一元素的零次或多次匹配 例如:“a*b”匹配“ab”,“aab”,“aaab”等
+	前一元素的一次或多次匹配 例如:“a+b”匹配“aab”,“aaab”等
{m,}	前一元素至少重复m次 例如:“a{2,}”匹配“aa”,“aaa”,“aaaa”等
{m,n}	前一元素至少重复m次,不超过n次 例如:“a{2,4}”匹配“aa”,“aaa”,“aaaa”
|	符号前或后的元素之一 例如:“foo|bar”匹配“boo”或者“bar”
^	字符串内起始位置 例如:“^foo”仅匹配起始处有“foo”的字符串
$	字符串内结束位置 例如:“bar\$”仅匹配结尾处有“bar”的字符串

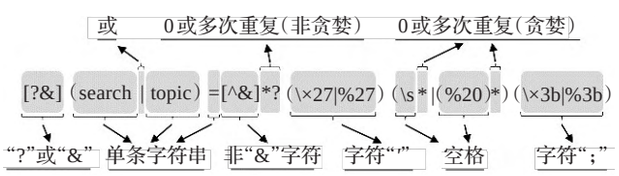


图1　Snort中用来检测Twiki非法远程调用的正则表达式规则

之,对于任何一个有穷自动机 $\mathcal{A}$,总存在一条正则表达式 r ,其表达的语义与 $\mathcal{A}$ 完全相同。非确定型有穷自动机(Nondeterministic FA,NFA)与确定型有穷自动机(Deterministic FA,DFA)是最常见的两种自动机,它们均由五元组 (Q,Σ,δ,q_0,F) 构成,如下:

Q :有限的状态集合,表示自动机的状态空间。

Σ :有限的输入字符集合,表示自动机接受的输入字符范围(对于网络流量检测,一般为ASCII中的256个字符)。

δ :状态转移函数, $Q\times\Sigma\rightarrow Q$,表示对于每一个输入字符,自动机的状态应该如何进行跳转。

$q_0\subseteq Q$:初始状态,表示自动机匹配前所处的状态。

$F\subseteq Q$:匹配状态的集合,表示跳转到自动机中这些状态时,发生了规则匹配。

在DFA中,状态转移函数 δ 在任意时刻对于输入字符仅返回一个确定的状态,即DFA中状态的跳转是唯一确定的。而对于NFA,状态转移函数 δ 会返回三种结果:一个状态,多个状态或者空。因此,NFA中的状态转移是非确定的。Thompson构造法[11]是常见的将正则表达式集合转换为NFA的算法。此外,McNaughton-Yamada构造法[16]是另一种将正则表达式转化为自动机的方法。构造后的NFA可以进行ε精简化,以消除冗余的ε状态和转移边[17]。NFA通过子集构造法[15]可被转化为DFA,而DFA可以进一步的最小化,消除多余状态和死状态。

图2和图3分别展示了规则“ab.*cd”对应的NFA和DFA结构图。图中,状态0为起始状态,双层圆圈为匹配状态(表示若该状态激活,则有规则被匹配)。状态之间的箭头表示状态转移,如果箭头上的字符为输入字符,则发生如箭头所示的状态跳转。例如在图2中,若当前状态1处于激活状态,输入字符是“b”,那么状态1将会沿着箭头跳转至状态3,即读入输入字符“b”后,状态3处于激活状态,状态1处于非激活状态。

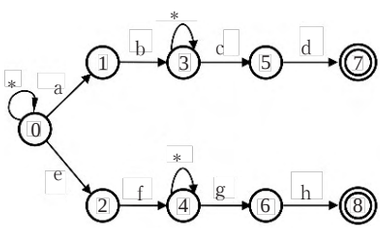


图2　规则“ab.*cd”对应的NFA

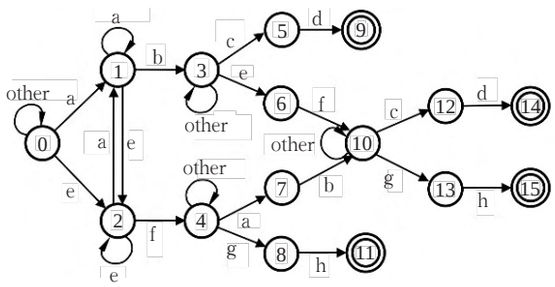


图3　规则“ab.*cd”对应的DFA

从图中可以直观地看出,NFA相对于DFA结构更简单。理论分析[18]表明,对于长度为 n 的正则表达式规则,NFA的空间复杂度为 $O(n)$ 。但是,NFA中可能出现多个状态同时激活的情况。例如,当图2中状态3处于激活状态时,若输入字符“c”,状态3会跳转到状态5,同时由于状态3有一个对于任何字符均指向自己的状态转移,因此状态3依旧处于激活状态。所以读入输入字符“c”后,状态3和状态5均处于激活状态。在最坏情况下,NFA中所有状态均处于激活状态,而每一个状态可能会跳转到其他所有的状态,因此对于长度为 n 的正则表达式规则,NFA在通用软件平台上的时间复杂度高达 $O(n^2)$ 。

在DFA中,所有状态的状态跳转是确定和唯一的。例如,若图3中状态3处于激活状态,输入字符是“c”,那么状态3将会确定地跳转到状态5,不会再激活其他状态。因此,DFA的时间复杂度为 $O(1)$,在匹配性能上相对于NFA有巨大的优势,逐渐成为深度检测中规则匹配的首选方案。然而,DFA优秀的时间性能是以

巨大的空间消耗为代价的。子集构造法实际上是遍历NFA中所有可能的状态组合,将每一种状态组合转化为DFA中的一个新状态,因此,对于长度为 n 的正则表达式规则,DFA的空间复杂度高达 $O(|\Sigma|^n)$ ($|\Sigma|$ 是输入字符集合中的元素个数)。在图2和图3的例子里,规则"ab.*cd"对应的NFA有9个状态,而DFA有16个状态,空间消耗增长了近一倍。当语义更加复杂,规则数目更大时,DFA所需的状态空间将指数级增长,甚至超过编译器的物理内存上限,发生"状态爆炸"。因此,DFA方法很难直接应用于大规模复杂规则集。

此外,子集构造法本身的时间复杂度也为指数级,由NFA构造DFA需要大量的预处理时间,使得DFA方法难以应对规则频繁更新的场景。

2.3 现阶段的挑战

正则表达式匹配技术经过数十年的探索与发展,已经有一定的成果与积累。然而,目前的匹配算法依然存在诸多待解决的问题。

(1)匹配速度

深度检测规则匹配引擎的处理速度一直是正则表达式规则匹配算法最主要的关注点之一。随着网络技术的不断发展,网络带宽已从传统的1 Gb/s、10 Gb/s,逐步提高到40 Gb/s、100 Gb/s,正迈向400 Gb/s大关。另一方面,软件定义网络(Software-Defined Networking,SDN)和网络功能虚拟化(Network Functions Virtualization,NFV)等技术逐渐兴起和成熟后,越来越多的网络功能与专有硬件设备解耦,采用通用的计算、存储、网络设备实现各种网络功能,降低成本,并且同时带来灵活部署、资源共享等多方面优势。然而,通用平台上的正则表达式匹配算法性能难以与FPGA、TCAM等专有硬件相比,匹配速度成为高性能网络流量检测与管理的瓶颈之一。

(2)内存占用

深度检测使用的正则表达式规则数目增长迅速。以开源入侵检测系统Snort为例,它从2003年开始使用正则表达式对匹配特征进行描述。到2006年,采用正则表达式的规则数目增加到1 131条。截止到2018年,这一数字已经快速上升到28 086条(根据snortrules-snapshot-29111规则集进行统计,该规则集发布于2018年1月16日)。许多实验[19-20]表明,对于较小规模规则集(编译后的DFA约有1万个状态),DFA内存占用约为10 MB,传统的规则匹配引擎能取得较高的匹配性能。但是,对于像Snort这样超过10 000条规则的大规模规则集,所对应的DFA状态数会超过1百万,内存占用达到数GB,超过了很多深度检测设备的内存空间上限。除了规则数目之外,为了描述丰富的匹配特征,规则的语义也变得越来越复杂。复杂的语义增大了自动机的处理难度,进一步加剧了匹配过程中的内存占用。

(3)更新能力

在SDN与NFV环境下,网络拓扑动态变化。传统的网络流量检测和管理设备被抽象为虚拟的网络功能,可以在网络拓扑中灵活实施和扩展。此外,为了及时应对突发网络威胁,安全规则需要快速部署和及时生效。例如,在思科的邮件安全应用中,检测规则的更新频率为3~5 min[21]。因此,网络流量的检测和管理规则动态性大大增加。然而,目前常见的基于DFA的规则匹配方法在规则数目较多时,预处理耗时太长,无法满足动态更新规则的网络管理需求。另一方面,基于NFA的规则匹配方法虽然预处理时间短,但是较低的匹配性能制约了NFA方法在网络流量检测与管理中(特别是虚拟网络环境下)的大范围应用。如何在规则预处理时间受限的情况下,实现快速的规则更新和高速的规则匹配,是正则表达式匹配方法所面临的新挑战。

3 研究现状分析

自21世纪初至今,正则表达式匹配问题的相关研究持续受到高度关注,涌现出了一大批优秀的研究成果。本章着眼于通用平台上的正则表达式匹配算法,按照压缩自动机空间、提升匹配速度、设计新型自动机以及规则拆分和分组四个方面,对代表性的正则表达式匹配算法进行归类、分析和总结。

3.1 压缩自动机空间

DFA处理速度上的优势使其成为规则匹配的首选方案。然而,DFA过高的空间复杂度是大规模规则匹配应用中不可回避的问题。目前学术界已有不少研究着眼于解决以DFA自动机空间消耗过大的问题[22-31]。通常来说,DFA使用状态转移表(State Transition Table,STT)表示自动机的结构,状态转移表可以看作一个二维的矩阵,矩阵的行表示DFA中的状态,矩阵的列表示输入字符。从不同的切入点入手,DFA空间压缩方法可以分为三类:字母表重编码、相似状态合并以及冗余转移边压缩。

在DFA的状态转移表中,如果存在多个不同的输入字符,所有的DFA状态在这些输入字符下状态跳转情况一致,那么这些输入字符可以看作是等价的。因此,可以用一个新的字符重新编码这些输入字符,从而达到压缩状态转移表列的目的[22-23]。如图4所示,在原始状态转移表中,输入字符a、c、d所在列完全相同,因此可以用一个新的输入字符a'重新编码a、c、d三个字符。同理,b和e可以由新字符b'替代,等等。字母表重新编码后,状态转移表的列数由8压缩至4,因此所占空间减少了50%。通过在查找状态转移表前使用一个专门的

索引表将原输入字符与重新编码后的输入字符对应,压缩后的状态转移表与原始状态转移表表示完全相同的DFA。Kong等人[24]进一步观察到,在多数情况下一些输入字符对于绝大多数DFA状态是等价的,仅有一小部分状态在这些输入字符下跳转情况不同。因此,Kong等人将整个状态集合划分成若干互不相交的子集,对于每一个子集使用不同的字母表重编码方式。在状态合并方面,Becchi等人[25]提出了一种将多个相同跳转目的地的状态合并为一个状态的方法。通过引入带标签的状态转移边,这些相似的状态可以合并为一个状态,从而大大减少DFA状态数。实验结果证明该方法与原始DFA状态转移表相比,可以节省90%的空间消耗。

	输入字符						
DFA状态	a	b	c	d	e	f	g
1	3	2	3	3	2	2	1
2	2	3	2	2	3	3	1
3	2	4	2	2	1	4	4
4	3	1	3	3	1	1	1

(a)原始状态转移表

	输入字符			
DFA状态	a'	b'	c'	d'
1	3	2	2	1
2	2	3	3	1
3	2	4	1	4
4	3	1	1	1

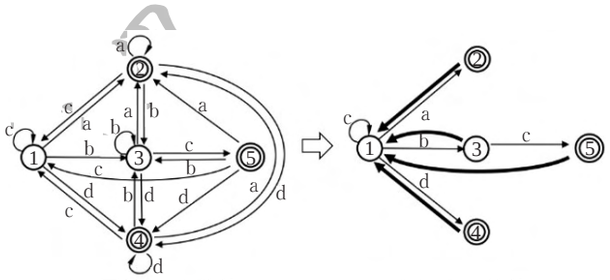
(b)字母表重编码后的状态转移表

图4　字母表重编码方法示例

字母表重编码和相似状态合并减少了DFA状态转移矩阵的列数或者行数,而冗余转移边压缩通过精简的表示方法,等同于减少了状态转移表中有效元素的个数。

D^{2}FA[26-27]是最经典的转移边压缩方法之一。图5表示了正则表达式规则"a+""b+c"和"c*d+"对应的原始DFA以及冗余转移边压缩后的D^{2}FA的示意图。三条规则对应的原始DFA结构如图5(a)所示,图5(b)表示压缩后的D^{2}FA结构,其中加粗箭头表示该状态的默认转移。图5(c)和图5(d)展示了两种方法的状态转移表。如果在D^{2}FA的状态转移表中无法找到相对应的状态转移,那么先按照默认转移跳转到下一状态,之后继续查找状态转移表,不断重复上述步骤,直到找到正确的下一跳状态为止。例如,当状态3处于激活状态,输入字符是"a"时,由于无法在D^{2}FA状态转移表找到相对应的下一跳状态,因此状态3先按照默认转移跳转至状态1,继续查找状态转移表,找到状态1在输入字符是"a"时的下一跳状态为状态2,因此状态3会跳转至状态2。在实践中,D^{2}FA最多可以取得95%的压缩率。然而,在最坏情况下,对于每个输入字符,D^{2}FA需要多次默认状态转移才能找到正确的状态,匹配性能无法得到保证。

许多方法对D^{2}FA作出了改进[28-31]。A-DFA[28]通过限制默认转移状态的方向,取得了确定的默认状态跳转次数上界。在最坏情况下,A-DFA中每个输入字符需要平均两次内存访问。然而,相比于D^{2}FA,A-DFA牺牲了



(a)原始DFA结构　(b)D^{2}FA结构

	输入字符				
DFA状态	a	b	c	d	
1	2	3	1	4	
2	2	3	1	4	
3	2	3	5	4	
4	2	3	1	4	
5	2	3	1	4	

(c)原始DFA状态转移表

	输入字符				
DFA状态	a	b	c	d	默认转移
1	2	3	1	4	
2					1
3			5		1
4					1
5					1

(d)D^{2}FA状态转移表

图5　D^{2}FA算法示例

一定的压缩率。OD^{2}FA[29]通过将源自同一NFA状态的多个DFA状态合并为一个超级状态,进一步压缩空间。Patel等人[30]提出一种"先最小化再联合"的框架,可以先对于每一条正则表达式构造出单独的D^{2}FA,然后将多个D^{2}FA合并为一个整合的D^{2}FA,构造起来更加灵活。δFA[31]关注于相邻状态间的冗余转移。对于任意状态,δFA仅储存与父状态(父状态定义该状态的上一跳状态)不同的状态转移。在实际状态跳转时,δFA通过将父状态的状态转移情况与该状态相对于父状态的差别相结合,即可得到该状态的状态转移情况。对于每一个输入字符,δFA平均仅需1次内存访问。但是,δFA的匹配速度提升依旧是以较低的压缩率作为代价的。

综上所述,DFA空间压缩方法实际上是在空间消耗和匹配时间之间取得一个权衡。几乎所有压缩方法都依赖于DFA的生成,因此并不能从根源上解决DFA状态爆炸问题。此外,这些方法均没有考虑预处理时间上的消耗。实验结果表明,在通用处理器平台上构造12条正则表达式对应的D^{2}FA需要超过71小时的时间;即使采用优化后的D^{2}FA合并方法,构造19条正则表达式对应的D^{2}FA依旧需要超过77分钟的时间[32]。因此,过长的预处理时间使得这些方法不适用于大规模规则集下规则频繁更新的动态网络场景。

3.2　提升匹配速度

近年来,一些研究利用自动机或者计算平台的特性,加速自动机的匹配速度[19,22-23,33-43]。

多步自动机通过在一个时间周期读入多个输入字符,理论上能够取得成倍的吞吐率[22-23]。图6展示了正则表达式规则"ab+[cd]e"对应的NFA以及两步NFA(2-NFA)。在图6(b)的2-NFA中,匹配引擎一次性读入两个待处理字符。如果两个输入字符为"ab",那么2-NFA中的状态2将被激活。如果之后的两个输入字符是"cd"或者"de",那么状态4将被激活,即发生了匹配。若

不考虑缓存命中等体系结构层面的优化处理，那么2-NFA的吞吐率相较于原始NFA将会提高一倍。类似地，已有工作提出了k步的k-NFA以及k-DFA的工作，并利用FPGA、TCAM等硬件平台的并行特性加速正则表达式规则匹配[19,33-36]。然而，基于多步的自动机加速方法获得的性能上的提升是以指数级增长的空间消耗为代价。对于字母表大小为$|\Sigma|$的原始自动机，k-NFA/k-DFA等价于定义在大小为$|\Sigma|^k$的字母表上的自动机。因此，只有当自动机规模足够小时，多步自动机才能取得较好的加速效果[20]。

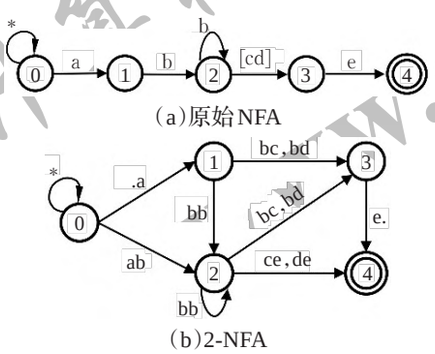


图6　规则“ab+[cd]e”对应的NFA以及2-NFA

利用多核平台进行数据并行匹配是在通用平台上提高正则表达式匹配速度的另一种常见方法[37-43]。为了解决正则表达式匹配对数据的强依赖性，传统的数据并行加速方法可以分为两类：一类方法是推测每一个数据块所对应的自动机的起始状态[37-38]，并由此状态进行状态的跳转。在前一个数据块完成匹配后，算法需要检查推测的自动机起始状态与前一数据块对应的自动机结束状态是否相同。若不相同，则这一数据块需要以前一数据块对应的自动机结束状态为新的起始状态，重新进行匹配，如图7(a)所示。当正则表达式规则规模较大时，所对应的自动机状态变多，这将导致推测的自动机起始状态与真实的起始状态不相符的可能性增大，故需要进行重新匹配的次数也大大增加。在实际数据匹配中，频繁地重新匹配将会降低整个深度检测系统的性能，吞吐能力无法得到保证。另一类方法是基于枚举的方法[39]，即枚举每一个数据块对应的自动机的所有状态作为起始状态，分别进行状态的跳转，如图7(b)所示。

对于每一个数据块，相应的计算核心独立计算所有状态对应的状态跳转，然后将不同计算核心得到的子结果按照顺序进行联合，得到最终的匹配结果。这种方法虽然不会有重新匹配的过程，但是额外计算开销较大。假设DFA的自动机状态数目为$|Q|$，m为输入数据的大小，n为并行计算的计算核心的数目，基于枚举的方法中匹配的时间复杂度为$O(|Q|\times m/n)$。这表明，当DFA自动机规模较大时，即$Q>>n$时，该方法与非并行匹配方法相比，并没有取得性能上的提升。

目前已有不少研究工作着眼于优化多核平台上的并行规则匹配。Mytkowicz等人[40]在基于枚举方法的基础上，利用SIMD指令减少枚举方法的额外开销，在16核的Intel Xeon处理器以及Snort单条规则集上能够取得最大8倍的加速效果。ParaRegex[41-42]通过设计“中间状态单元”的数据结构，以比特形式记录各数据块的匹配历史信息，并结合DFA匹配过程中的“状态汇聚”现象，进一步减少基于枚举方法的计算复杂度。实验表明，在大规模正则表达式规则集下，通过利用8核平台实现数据并行的规则匹配，能取得4至6倍左右的匹配加速，同样也能够并行化D^2FA等DFA衍生方法。Jiang等人[43]提出了一种将基于推测的方法与基于枚举的方法相结合的新型并行化匹配方法。与仅计算一个状态的基于推测方法以及需计算所有状态的基于枚举方法不同，作者提出的算法仅推测若干个最有可能的起始状态，既能降低基于推测方法的频繁重匹配概率，又能减少基于枚举方法的额外计算开销。实验表明，在60核的Intel Xeon Phi处理器上，该方法能够取得高达95倍的加速性能。

综上所述，着眼于提升自动机匹配速度的研究往往以更多的内存消耗或计算量为代价，这一类工作通常与自动机压缩方法相结合，共同实现高性能的正则表达式匹配。

3.3　设计新型自动机

一些研究通过构造不同于NFA和DFA的新型自动机，试图解决NFA匹配速度过慢和DFA状态空间爆炸的问题[44-51]。

Becchi等人[44]观察到，DFA中的状态爆炸主要源自正则表达式中的“.*”和有重复次数限制的字符集合。

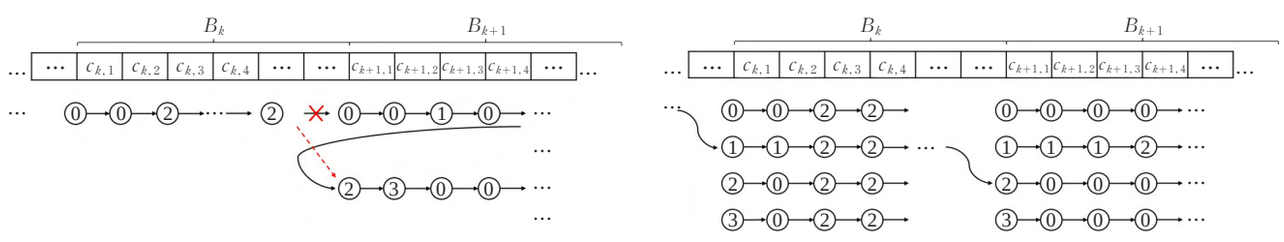


图7　两类数据并行的正则表达式匹配算法

因此,在由NFA构造DFA的过程中,在遇到".*"或者重复次数限制的字符集合中的第一个字符时,子集构造法会停止构建。因此,构造过程得到的结果是一个由头部DFA和若干尾部NFA共同构成的混合FA(Hybrid-FA),其中DFA与NFA之间通过特殊的"边界状态"相连,如图8所示。在匹配过程中,头部DFA中始终有一个状态处于激活状态;当且仅当"边界状态"被激活时,尾部NFA才会被激活。作者说明,由于绝大部分网络流量属于正常流量,头部DFA可以起到一个预过滤的作用。然而,攻击者可以构造一些恶意流量,使得Hybrid-FA的尾部NFA中状态被多次激活。一旦陷入尾部NFA,Hybrid-FA的性能会出现严重下降。因此,实际匹配中Hybrid-FA的性能无法得到保证。

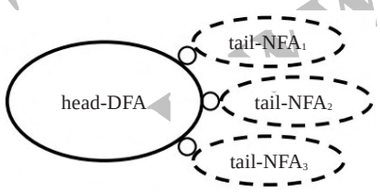


图8　Hybrid-FA算法结构示例

Yang等人[45]证明状态爆炸是由冲突的NFA状态所引起,由此提出将这些冲突的NFA状态划分为若干子集,并将每个子集编译成DFA,这些DFA共同构成了半确定型FA(Semi-deterministic FA,SFA)。SFA实际上将单个NFA的时间复杂度转为多个DFA的时间复杂度。当NFA有大量的冲突状态对时,SFA构造得到的DFA数量庞大,造成匹配性能的严重损失。此外,SFA判断NFA状态冲突的算法过于复杂,难以应用于大规模规则集。TFA[46](Tunable FA,TFA)是另一种介于DFA与NFA之间的新型自动机。与DFA使用一个状态、NFA使用一系列状态的集合表示当前匹配情况不同,TFA使用确定上界的b个状态表示当前匹配情况。每一个TFA状态可以看作是一系列NFA状态的组合,通过子集划分的方法,可以将每一个DFA状态映射为对应的若干TFA状态的组合。在匹配过程中,最多有b个TFA状态激活,因此这种方法可以保证最坏情况下的性能。然而,TFA构造过程中使用的子集划分方法是NP难的问题,预处理时间较长。此外,TFA的构造依然依赖于DFA的生成,无法从根本上解决状态爆炸的问题。

此外,Kumar等人[47]提出一种基于历史信息的FA,即H-FA(History based FA)。当遇到".*"、带重复次数字符集等会引起DFA状态膨胀的语义时,H-FA通过附加一些带有辅助变量的状态,记录这些特殊的事件。在匹配过程中,辅助变量会进行更新,通过辅助变量不同的值和DFA激活的状态替代原DFA中因".*"、带重复次数字符集等语义产生的DFA状态,从而在实现等价匹配的前提下,减少了DFA状态数。类似地,XFA[48-49](eXtended FA)通过引入形式化的模型,系统性地将辅助变量与操作指令统一起来。相对于H-FA,XFA中任一状态对于给定的输入字符仅需一次状态转移。而且,XFA可由单条正则表达式对应的XFA整合构造而得。JFA[50](Jump-FA)针对XFA无法处理部分类型正则表达式的缺陷作出了改进。JFA通过在状态转移上附着辅助标签保证自动机功能的完全等价,同时通过标签转发、标签构造新型自动机。切分等优化方法减少自动机的复杂度。实验结果表明,与H-FA相比,JFA能在最坏情况下取得更好的性能。与XFA相比,JFA能够支持更多类型的正则表达式形式。

以上基于新型自动机的方法需要深入分析自动机状态间或者规则语义本身的关系,预处理时间较长。实验表明,对于1 000多条正则表达式规则集,构造XFA需要2.5天至25天的时间[48]。此外,这些方法往往离不开人工干预,需要对自动机的状态转移或者正则表达式规则做人工标注等操作,当规则变动时难以及时处理。

为了解决规则频繁更新场景下的匹配问题,Fu等人[51]提出了支持规则快速预处理的BFA(Bit based FA)匹配引擎。BFA使用比特向量编码自动机中的状态和状态转移函数,如图9所示。BFA匹配阶段中,状态跳转过程转化为比特向量与比特矩阵的布尔矩阵乘法。例如,当原NFA中状态0和状态2处于激活状态,那么当输入字符是c时,状态跳转计算如下:

$$[1\,0\,1\,0\,0]\times\begin{bmatrix}1&0&0&0&0\\0&0&0&0&0\\0&0&1&1&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}=[1\,0\,1\,1\,0]$$

该结果表明读入字符c后,状态0、状态2和状态3将被激活。同时,该计算过程可以利用商用CPU中内置的比特操作及并行指令集进行优化,并通过位图压缩技术减少了BFA中状态转移表所占空间的大小。基于真实流量和规则的实验表明,BFA的规则预处理时间和内存占用远优于DFA方法,匹配速度较NFA方法高10至100倍。在对引擎更新能力的评价上,BFA与DFA等

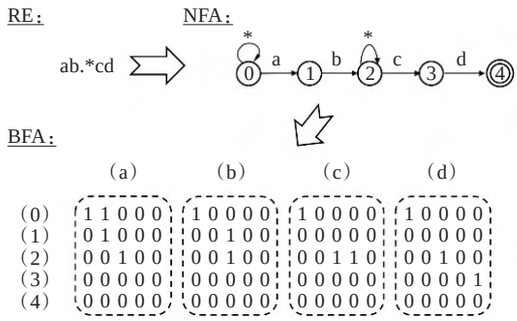


图9　支持规则快速预处理的BFA匹配引擎示意图

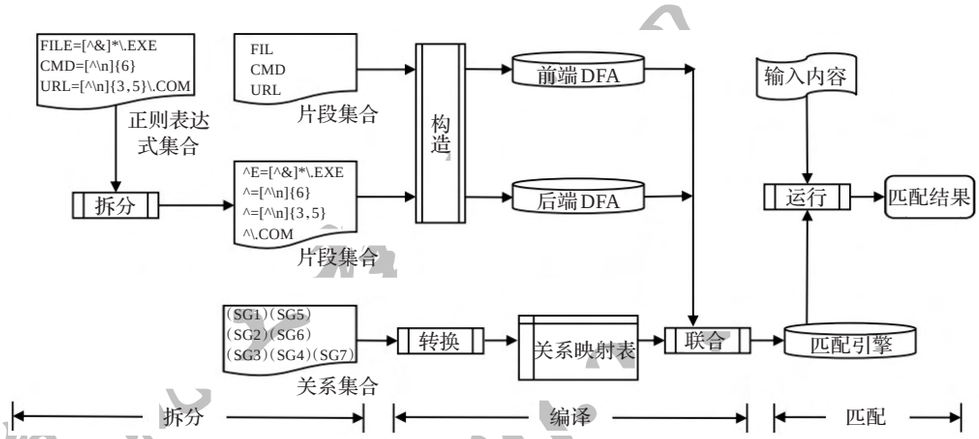


图10　PaCC算法流程示例

已有方法相比，有5倍至20倍的提高。此外，BFA在多核平台上具有良好的扩展性。与传统自动机方法相比，BFA获得了更好的预处理时间、匹配速度和内存占用间的权衡，更适用于动态网络场景。

3.4　规则拆分和分组

近年来，一些研究直接从正则表达式规则本身语义出发，通过对规则集进行处理，解决现有正则表达式匹配方法存在的问题[18,52-60]。Wang等人[52-53]提出了一种正则表达式拆分的方法PaCC。PaCC首先总结了可能导致DFA状态爆炸的若干种正则表达式语义类型，然后针对每一种语义类型提出相应的拆分方法。拆分后的前缀和后缀分别放入两个不同的集合中，并分别编译为DFA。后缀集合中的规则通过“锚定”符号，可以消除片段之间的语义交叠，从而减少DFA的空间消耗。此外，PaCC通过关系映射表保持规则片段间的语义联系，从而使得拆分后的规则集与原始规则集表达完全等价的语义。PaCC的匹配流程如图10所示。作者的实验证明，PaCC可以获得堪比NFA的空间消耗和堪比DFA的匹配速度。

规则分组通过将正则表达式集合分划为若干子集合，对各个子集合分别构造DFA，可有效解决DFA状态爆炸的问题。Becchi等人[54]在构造DFA时，若DFA所需的空间超过了预先设置的上限，则对规则集进行二等分，对两部分分别构造DFA。重复此步骤直至所有的规则均能生成DFA。Yu等人[18]首先定义了正则表达式之间的“冲突”程度，即当两条正则表达式编译为一个DFA后，若其状态总数大于两条正则表达式单独编译为DFA的状态数之和，那么这两条正则表达式互相“冲突”。基于此，Yu等人提出了一种贪婪的启发式分组方法，即当划分一个新分组时，选择与组中已有正则表达式冲突最少的规则加入到此组。Rohrer等人[55]基于Yu的分组方法，量化了正则表达式间的“冲突”程度。通过定义正则表达式之间的“距离”，Rohrer等人构造了无向带权图（如图11所示），并将规则分组问题转化为集合最大割问题，通过线性规划、启发式和模拟退火等多种方式进行了求解。Liu等人在文献[56]中用最大 k 割问题求解方法解决规则的最优 k 分组问题，并证明了所提出算法的结果不会差于最大 k 割问题最优解的 $(1-1/k)$ 倍。之后在文献[57]中，Liu等人通过分析NFA状态与DFA状态之间的关系，提出了一种DFA状态近似估计方法，并基于DFA状态数预估设计了新的正则表达式规则分组方法。

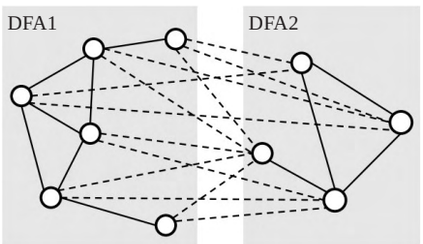


图11　规则分组转化为最大割问题

Fu等人[58-59]认为，规则分组问题并不能仅仅把分组后总的DFA状态数或者分组数目当作唯一的优化目标，分组问题的优化目标需要与实际需求相结合。不同的分组分案的匹配时间复杂度和空间复杂度如表2所示。当可供规则匹配引擎使用的计算核心数目确定的情况下，Fu等人提出了分组算法Reevo，首先对规则分组方案进行数值编码，利用遗传算法中的选择、交叉、变异的进化步骤，迭代求得近似最优的分组结果。图12给出了将6条正则表达式分为3组的例子。染色体长度为6，6个节点分别对应6条正则表达式 r_1 至 r_6，节点上的数值均在1至3之间。算法的初始化即将各条正则表达式随机分入某一组中。如图12(a)中第1行染色体所

表2　对于 m 条平均长度为 n 的正则表达式，不同的规则处理方法的匹配时间复杂度和空间复杂度

处理方式	时间复杂度	空间复杂度
每条规则单独编译为DFA	$O(m)$	$O(m\times\lvert\Sigma\rvert^n)$
将规则分为 k 组，每组编译为一个DFA	$O(k)$	$O(k\times\lvert\Sigma\rvert^{\frac{n\times m}{k}})$
所有规则编译为同一个DFA	$O(1)$	$O(\lvert\Sigma\rvert^{n\times m})$

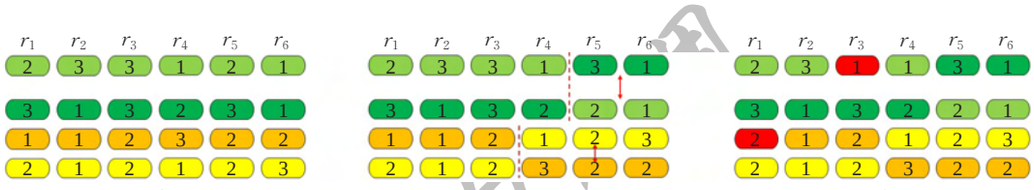


图12　Reevo规则分组算法中的编码、交叉和变异

表3　正则表达式匹配方法研究方向分类

研究方向	主要原理	代表研究工作
压缩自动机空间	通过字母表重编码、相似状态合并以及冗余转移边压缩减少DFA状态转移表的空间占用	D^2FA[26-27], A-DFA[28], OD^2FA[29], δFA[31]
提升匹配速度	利用自动机或者计算平台的特性,加速正则表达式匹配中自动机的跳转速度	*k*-NFA/*k*-DFA[22-23], ParaRegex[41-42]
设计新型自动机	构造不同于NFA和DFA的新型自动机,以解决NFA匹配速度过慢和DFA状态空间爆炸的问题	Hybrid-FA[44], SFA[45], XFA[48-49], BFA[51]
规则拆分和分组	从正则表达式规则本身语义出发,通过对规则集进行处理,解决现有正则表达式匹配方法存在的问题	PaCC[52-53], mDFA[57], Reevo[58-59]

示,r_4 与 r_6 对应节点的数值为1,表示规则 r_4 与 r_6 在初始阶段被分为第1组。类似地,r_1 与 r_5 在初始阶段被分为第2组,r_2 与 r_3 被分为第3组。如此,该染色体[2 3 3 1 2 1]即代表了一种分组方案。同样,可以产生另外3种初始分组方案,如第2,3,4行所示。编码后,各染色体进行评价、选择、交叉和变异,不断迭代直至满足停止条件。而另一种场景下,正则表达式规则分组后每组状态数目(存储空间)上限是确定的,在满足空间约束的条件下,首先需要保证每个分组的DFA能够生成,同时需要将正则表达式规则集合划分成尽可能少的组。针对这一问题,Fu等人提出了分组算法Reant,依照预先计算的概率归并正则表达式,并引入蚁群算法中"信息素"的概念,通过多次迭代得到最终分组方案。基于真实规则集的实验评测结果表明,通过Reevo和Reant算法对大规模规则集分组后,内存消耗比已有方法减少了约45%,分组数目比方法减少了约25%,可以有效解决大规模规则集内存消耗过大的问题。

以上基于智能优化的分组算法均需要大量的迭代计算。在对规则分组时间要求严格的场景下,Fu等人[60]提出了一种快速分组算法,在估算正则表达式规则间相似度的基础上,可以由 m 条正则表达式规则构造一个由 m 个节点和 $m\times(m-1)$ 条边构成的无向图,进而通过谱聚类[61]的思想对这 m 个节点进行初始分组。实验结果表明,该方法在与对比方法效果相似的条件下,可以减少二分之一至三分之二的计算分组方案所需时间。同时,该方法亦可作为智能优化分组算法Reevo和Reant的初始解方案,加速分组算法的迭代过程。

正则表达式规则拆分和分组方法从规则本身出发,解决大规模规则匹配过程中自动机内存占用过大的问题,与现有的其他方法相正交,可以作为常规方法的有效补充。

综上所述,正则表达式匹配方法研究方向可按表3所示进行总结分类。

4　实验评测及分析

针对2.3节提出的现阶段正则表达式匹配算法面临的挑战,本章从匹配速度、内存占用以及预处理时间三方面详细评测几种经典的正则表达式规则匹配算法。

实验采用Intel Core i7-4790处理器,配有8 GB内存以及Ubuntu 16.04系统。实验中评测了NFA,DFA,Hybrid-FA[44],D^2FA[26],BFA[51]以及规则分组算法[57](简称为mDFA),等匹配算法。实验中使用了6个不同的规则集,如表4所示。其中,两个规模较小的规则集snort1和snort2来自开源入侵检测和防御系统Snort[13],bro规则集来自网络安全检测软件Bro[62],tcp规则集来自于正则表达式处理引擎Regular Expression Processor[63]。dotstar和range为两个人工生成的规则集,其中dotstar规则集包含300条带有".*"的正则表达式规则,range规则集包含300条带有范围语义的正则表达式规则。这些人工生成的规则集处理起来更为复杂,对规则匹配算法的挑战更大。

表4　实验中所使用的规则集

规则集	snort1	snort2	bro	tcp	dotstar	range
规则数目	24	34	217	733	300	300
复杂程度	低	低	中	高	高	高

4.1　匹配速度

实验中,从校园网抓取了8 MB大小的真实流量文件,作为各匹配算法的输入数据。从表5中可以得知,DFA在snort1、snort2以及bro规则集中均取得了最快的匹配速度。D^2FA与DFA相比,需要首先访问默认的跳转状态,然后找到正确的下一跳转状态。因此,D^2FA匹

配速度要慢于DFA。在所有引擎中,NFA的匹配速度最慢,这可以归咎于状态跳转中不确定的内存访问。在Hybrid-FA中,由于绝大多数的非恶意流量能够被前端DFA过滤,仅有较少部分的流量需要被后端NFA检测,所以,Hybrid-FA的匹配速度要高于NFA。

表5　不同匹配算法在不同规则集上的匹配时间　s

匹配方法	snort1	snort2	bro	tcp	dotstar	range
NFA	16.897	16.950	122.406	239.075	148.933	71.589
DFA	0.072	0.074	0.109	\	\	\
mDFA	\	\	\	1.577	1.194	0.361
Hybrid-FA	0.699	0.849	2.044	\	\	\
D^2FA	1.163	1.172	0.956	\	\	\
BFA	1.521	1.512	3.620	2.080	2.077	2.053

实验中,对于规模较大或者较复杂的规则集,DFA、D^2FA和Hybrid-FA均不能生成对应的数据结构。经过分析,发现其原因是在数小时的计算之后,生成的自动机的内存消耗超过了实验平台可用内存的上限。为了处理这些规则集,mDFA通过分组的方法产生多个DFA,这些DFA经过依次匹配后,仍取得了不错的匹配速度。BFA能够处理实验中所有规模的正则表达式规则集,其匹配速度稍慢于Hybrid-FA、D^2FA以及mDFA,比NFA要快10到100倍。

4.2　内存占用

对比实验中,用不同算法编译各测试规则集,生成相应的匹配引擎,衡量其占用内存大小。

如表6所示,由于NFA使用的数据结构最为紧凑,因此NFA的内存占用总体最小。DFA所需的内存空间最大,平均约为NFA的35倍。相比于DFA,D^2FA通过冗余状态转移边压缩的方式,有效地压缩了自动机的内存占用。对于bro规则,其所需的内存空间甚至比NFA还要小。Hybrid-FA与BFA的内存消耗均介于NFA与DFA之间。

对于复杂规则集(tcp、dotstar与range),BFA仅仅需要约20 MB至30 MB的内存消耗,而DFA、Hybrid-FA和D^2FA的内存消耗均超过了实验平台的物理内存上限。mDFA通过规则分组的方式,成功在实验平台上生成了较复杂规则集的DFA,其内存消耗为400 MB至600 MB。

4.3　预处理时间

从表7中可以看到,NFA的预处理时间在所有匹配引擎中是最快的。这是由于其他所有引擎的构建均以NFA为基础,因此需要额外的预处理时间。DFA与NFA相比,需要平均100倍以上的预处理时间,这是因为由NFA构造DFA的子集构造法需要穷举NFA中所有可能的状态组合,非常耗时。当规则集规模较大、语义复杂时,情况更为严重。

Hybrid-FA的预处理时间介于NFA和DFA之间,因为Hybrid-FA实际上构建了一个由头部DFA和若干尾部NFA共同组成的混合结构,预处理时间相对DFA更少。由于D^2FA旨在压缩DFA中冗余的状态转移边,需要先生成DFA后再对所有的状态转移边进行分析,生成默认跳转状态,因此,D^2FA需要比DFA更多的预处理时间。BFA与NFA相比,仅需额外20%至50%的预处理时间。这是由于BFA对NFA进行比特编码,相对于子集构造法复杂程度大大降低。而且,BFA与NFA一样,能够成功处理实验中所有大小和复杂程度的规则集。对于较大规模规则集,mDFA由于需要计算每两条正则表达式规则间的“冲突”关系,因此其分组过程耗时高达4至10小时。

4.4　分析与讨论

由以上实验分析可知,DFA算法在自动机能够生成的前提下,取得了最快的匹配速度;mDFA通过规则分组,能够有效解决大规模规则集下DFA状态爆炸的问

表6　不同匹配算法在不同规则集上的内存占用　KB

匹配方法	snort1	snort2	bro	tcp	dotstar	range
NFA	162.520	250.882	605.470	4 035.996	3 165.721	3 314.318
DFA	8 835.040	9 988.096	6 689.792	\	\	\
mDFA	\	\	\	485 840.786	609 587.440	620 601.645
Hybrid-FA	1 213.421	1 261.441	1 713.247	\	\	\
D^2FA	264.122	310.149	172.956	\	\	\
BFA	1 203.646	1 843.654	4 413.894	30 069.300	23 212.102	24 041.348

表7　不同匹配算法在不同规则集上的预处理时间　s

匹配方法	snort1	snort2	bro	tcp	dotstar	range
NFA	0.127	0.153	0.320	52.244	3.052	1.827
DFA	18.535	26.357	38.985	\	\	\
mDFA	\	\	\	34 649.657	15 859.416	20 965.463
Hybrid-FA	2.494	3.318	18.024	\	\	\
D^2FA	20.398	30.247	39.917	\	\	\
BFA	0.168	0.216	0.493	60.719	4.131	2.593

题。然而，DFA以及mDFA均需要大量的预处理时间。NFA算法的预处理时间最快，且内存空间消耗最少。但是，NFA的匹配性能有限。

Hybrid-FA以及D²FA在小规模规则集上取得较好的空间压缩效果，但是依然难以处理大规模正则表达式规则集。BFA的较短的预处理时间内能够取得较高的匹配速度，匹配“性价比”最高。进一步地，分别以匹配速度、内存空间消耗以及预处理速度为轴，在图13中刻画了以上几种不同匹配算法的位置。

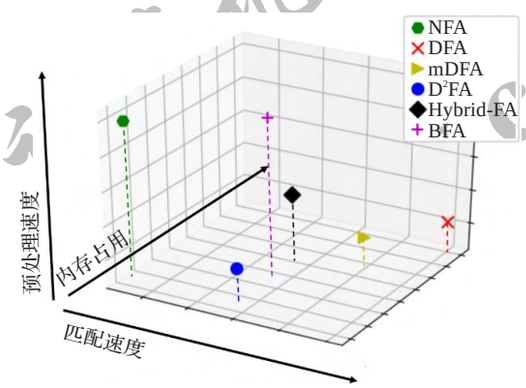


图13　不同匹配方法的匹配速度、内存占用以及预处理速度比较

对于实际场景，针对不同的匹配需求，可以按照以下原则选择合适的正则表达式规则匹配算法：

(1)如果待匹配规则十分简单，并且对匹配性能要求不高，那么NFA方法由于其构造快捷以及内存占用少的性质，成为最高效的匹配方法。实际上，以grep，awk为代表的许多匹配工具选择NFA作为默认的正则匹配引擎。

(2)如果对匹配性能要求很高，且能够容忍较长时间的预处理时间，那么DFA方法无疑是最合适的匹配算法。而对于大规模正则表达式规则集，规则分组可以有效地解决状态空间“爆炸”的问题。

(3)如果对规则预处理时间有严苛要求，同时希望能够在较短的预处理时间内取得尽可能高的匹配速度，那么BFA方法获得了更好的预处理时间、匹配速度和内存占用间的权衡，更适用于这种动态网络场景。

5　总结和展望

正则表达式匹配算法是深度检测等网络流量检测和管理的核心技术之一。网络流量的持续增长、规则数目的不断增多以及网络结构的日趋灵活给现有的正则表达式匹配算法带来了严峻的挑战。本文介绍了正则表达式匹配算法的背景，从空间压缩、匹配加速、新型自动机设计以及规则拆分和分组四个角度对学术界具有影响力的基于通用平台的研究成果进行分类总结，并通过实际测试比较了几种具有代表性的方法的匹配速度、内存占用以及预处理速度等指标，分析了各方法的优缺点以及适用场景。

正则表达式匹配算法相关研究方向上的未来工作可以从如下几方面开展：

(1)研究自适应的规则匹配算法。NFA、DFA等已有的匹配算法关注于解决某一场景或者某些场景下的规则匹配问题，然而规则匹配的实际需求随着时间或者环境变化而有所不同。如果规则匹配引擎能够根据应用目标自动选择合适的匹配方法，那么其综合效能无疑会更好。因此，研究能够不断优化的自适应深度检测引擎具有重要的研究价值。

(2)研究加密场景下的规则匹配算法。大数据时代，数据隐私成为了人们普遍关注的问题。目前的规则匹配算法需要接触原始数据，难以应对加密流量场景。因此，如何在隐私保护的角度下，实现加密网流的规则匹配也将是未来工作的热点和难点之一。

(3)研究网络技术演进中正则表达式匹配算法的新需求。随着物联网、边缘计算等新兴网络技术的发展与成熟，正则表达式匹配算法的需求更加多样化，未来工作需要考虑网络发展过程中出现的新需求与新挑战。例如，衡量规则匹配算法的能耗比，研究能耗更低的规则匹配方法，可有效满足低功耗设备的需求。

参考文献：

[1] Internet usage statistics-the internet big picture[EB/OL]. https://www.internetworldstats.com/stats.htm.

[2] 第41次《中国互联网络发展状况统计报告》[EB/OL]. http://www.cnnic.net.cn/hlwfzyj/hlwxzbg/hlwtjbg/201803/t20180305_70249.htm.

[3] Cisco visual networking index:forecast and methodology, 2016—2021[EB/OL].https://www.cbsnews.com/news/wannacry-ransomware-attacks-wannacry-virus-losses.

[4] Memcached ddos:the biggest,baddest denial of service attacker yet[EB/OL].http://www.zdnet.com/article/memcached-ddos-the-biggest-baddest-denial-of-service-attacker-yet.

[5] 尹浩，姜泽勋，李军.联接云际：云际网络互联的协同服务机制[J].中国计算机学会通讯，2017(3).

[6] Knuth D E,Morris J,James H,et al.Fast pattern matching in strings[J].SIAM Journal on Computing,1977,6(2):323-350.

[7] Boyer R S,Moore J S.A fast string searching algorithm[J]. Communications of the ACM,1977,20(10):762-772.

[8] Aho A V,Corasick M J.Efficient string matching:an aid to bibliographic search[J].Communications of the ACM,1975,18(6):333-340.

[9] Wu S,Manber U.A fast algorithm for multi-pattern

searching[Z].Tucson,AZ:University of Arizona,1994.

[10] Kleene S C.Representation of events in nerve nets and finite automata[R].Santa Monica,CA:RAND Project Air Force,1951.

[11] Thompson K.Programming techniques:regular expression search algorithm[J].Communications of the ACM,1968,11(6):419-422.

[12] Ritchie D M.The evolution of the UNIX time-sharing system[M]//Language design and programming methodology.[S.l.]:Springer,1980:25-35.

[13] Snort[EB/OL].https://www.snort.org.

[14] Twiki-the open source enterprise wiki and web application platform[EB/OL].http://twiki.org.

[15] Hopcroft J E,Motwani R,Ullman J D.Introduction to automata theory,languages,and computation[J].ACM SIGACT News,2001,32(1):60-65.

[16] Mcnaughton R,Yamada H.Regular expressions and state graphs for automata[J].IRE Transactions on Electronic Computers,1960(1):39-47.

[17] Hopcroft J.An n log n algorithm for minimizing states in a finite automaton[C]//Theory of Machines and Computations,1971:189-196.

[18] Yu F,Chen Z,Diao Y,et al.Fast and memory-efficient regular expression matching for deep packet inspection[C]//Proceedings of the 2006 ACM/IEEE Symposium on Architecture for Networking and Communications Systems,2006:93-102.

[19] Chen X,Jones B,Becchi M,et al.Picking pesky parameters:optimizing regular expression matching in practice[J].IEEE Transactions on Parallel and Distributed Systems,2016,27(5):1430-1442.

[20] Xu C,Chen S,Su J,et al.A survey on regular expression matching for deep packet inspection:applications,algorithms,and hardware platforms[J].IEEE Communications Surveys and Tutorials,2016,18(4):2991-3029.

[21] Cisco email security appliance data sheet[EB/OL].https://www.cisco.com/c/en/us/products/collateral/security/email-security-appliance/data-sheet-c78-729751.html.

[22] Becchi M,Crowley P.Efficient regular expression evaluation:theory to practice[C]//Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems,2008:50-59.

[23] Brodie B C,Taylor D E,Cytron R K.A scalable architecture for high-throughput regular-expression pattern matching[J].ACM SIGARCH Computer Architecture News,2006,34(2):191-202.

[24] Kong S,Smith R,Estan C.Efficient signature matching with multiple alphabet compression tables[C]//Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks,2008:1.

[25] Becchi M,Cadambi S.Memory-efficient regular expression search using state merging[C]//INFOCOM 2007-26th IEEE International Conference on Computer Communications,2007:1064-1072.

[26] Kumar S,Dharmapurikar S,Yu F,et al.Algorithms to accelerate multiple regular expressions matching for deep packet inspection[C]//ACM SIGCOMM 2006 Conference on Applications,Technologies,Architectures,and Protocols for Computer Communications,2006:339-350.

[27] Becchi M,Crowley P.An improved algorithm to accelerate regular expression evaluation[C]//Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems,2007:145-154.

[28] Becchi M,Crowley P.A-DFA:a time-and space-efficient DFA compression algorithm for fast regular expression evaluation[J].ACM Transactions on Architecture and Code Optimization(TACO),2013,10(1):4.

[29] Liu A X,Torng E.An overlay automata approach to regular expression matching[C]//2014 Proceedings IEEE INFOCOM,2014:952-960.

[30] Patel J,Liu A X,Torng E.Bypassing space explosion in high-speed regular expression matching[J].IEEE/ACM Transactions on Networking(TON),2014,22(6):1701-1714.

[31] Ficara D,Giordano S,Procissi G,et al.An improved DFA for fast regular expression matching[J].ACM SIGCOMM Computer Communication Review,2008,38(5):29-40.

[32] Patel J,Liu A X,Torng E.Bypassing space explosion in regular expression matching for network intrusion detection and prevention systems[C]//Proceedings of the 19th Annual Network and Distributed System Security Symposium,2012.

[33] Cho Y H,Navab S,Mangione-Smith W H.Specialized hardware for deep network packet filtering[C]//International Conference on Field Programmable Logic and Applications,2002:452-461.

[34] Yamagaki N,Sidhu R,Kamiya S.High-speed regular expression matching engine using multi-character NFA[C]//International Conference on Field Programmable Logic and Applications,2008:131-136.

[35] Nakahara H,Sasao T,Matsuura M.A regular expression matching circuit based on a modular non-deterministic finite automaton with multi-character transition[C]//Proc 16th Workshop on Synthesis And System Integration of Mixed Information Technologies,2010:359-364.

[36] Yang J,Jiang L,Tang Q,et al.PiDFA:a practical multi-stride regular expression matching engine based on FPGA[C]//2016 IEEE International Conference on Com-

munications(ICC),2016:1-7.

[37] Luchaup D,Smith R,Estan C,et al.Multi-byte regular expression matching with speculation[C]//International Workshop on Recent Advances in Intrusion Detection,2009:284-303.

[38] Luchaup D,Smith R,Estan C,et al.Speculative parallel pattern matching[J].IEEE Transactions on Information Forensics and Security,2011,6(2):438-451.

[39] Holub J,Štekr S.On parallel implementations of deterministic finite automata[C]//International Conference on Implementation and Application of Automata,2009:54-64.

[40] Mytkowicz T,Musuvathi M,Schulte W.Data-parallel finite-state machines[J].ACM SIGARCH Computer Architecture News,2014,42(1):529-542.

[41] Fu Z,Liu Z,Li J.ParaRegex:towards fast regular expression matching in parallel[C]//2016 ACM/IEEE Symposium on Architectures for Networking and Communications Systems(ANCS),2016:113-114.

[42] Fu Z,Liu Z,Li J.Efficient parallelization of regular expression matching for deep inspection[C]//2017 26th International Conference on Computer Communication and Networks(ICCCN),2017:1-9.

[43] Jiang P,Agrawal G.Combining SIMD and many/multi-core parallelism for finite state machines with enumerative speculation[J].ACM SIGPLAN Notices,2017,52(8):179-191.

[44] Becchi M,Crowley P.A hybrid finite automaton for practical deep packet inspection[C]//Proceedings of the 2007 ACM CoNEXT Conference,2007:1.

[45] Yang Y H E,Prasanna V K.Space-time tradeoff in regular expression matching with semi-deterministic finite automata[C]//2011 Proceedings IEEE INFOCOM,2011:1853-1861.

[46] Xu Y,Jiang J,Wei R,et al.TFA:a tunable finite automaton for pattern matching in network intrusion detection systems[J].IEEE Journal on Selected Areas in Communications,2014,32(10):1810-1821.

[47] Kumar S,Chandrasekaran B,Turner J,et al.Curing regular expressions matching algorithms from insomnia,amnesia,and acalculia[C]//Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems,2007:155-164.

[48] Smith R,Estan C,Jha S.XFA:faster signature matching with extended automata[C]//2008 IEEE Symposium on Security and Privacy,2008:187-201.

[49] Smith R,Estan C,Jha S,et al.Deflating the big bang:fast and scalable deep packet inspection with extended finite automata[J].ACM SIGCOMM Computer Communication Review,2008,38(4):207-218.

[50] Yu X,Lin B,Becchi M.Revisiting state blow-up:automatically building augmented-FA while preserving functional equivalence[J].IEEE Journal on Selected Areas in Communications,2014,32(10):1822-1833.

[51] Fu Z,Zhou S,Li J.bitFA:a novel data structure for fast and update-friendly regular expression matching[C]//Proceedings of the SIGCOMM Posters and Demos,2017:130-132.

[52] Wang K,Fu Z,Hu X,et al.Practical regular expression matching free of scalability and performance barriers[J].Computer Communications,2014,54:97-119.

[53] Wang K,Li J.FREME:a pattern partition based engine for fast and scalable regular expression matching in practice[J].Journal of Network and Computer Applications,2015,55:154-169.

[54] Becchi M,Franklin M,Crowley P.A workload for evaluating deep packet inspection architectures[C]//IEEE International Symposium on Workload Characterization(IISWC),2008:79-89.

[55] Rohrer J,Atasu K,Van Lunteren J,et al.Memory-efficient distribution of regular expressions for fast deep packet inspection[C]//Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis,2009:147-154.

[56] 柳厅文,孙永,卜东波,等.正则表达式分组的$1/(1-1/k)$-近似算法[J].软件学报,2012,23(9):2261-2272.

[57] Liu T,Liu A X,Shi J,et al.Towards fast and optimal grouping of regular expressions via DFA size estimation[J].IEEE Journal on Selected Areas in Communications,2014,32(10):1797-1809.

[58] Fu Z,Wang K,Cai L,et al.Intelligent grouping algorithms for regular expressions in deep inspection[C]//2014 23rd International Conference on Computer Communication and Networks(ICCCN),2014:1-8.

[59] Fu Z,Wang K,Cai L,et al.Intelligent and efficient grouping algorithms for large-scale regular expressions[J].Computers & Electrical Engineering,2018,67:223-234.

[60] Fu Z,Li J.Spectral clustering based regular expression grouping[C]//Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems,2014:243-244.

[61] Ng A Y,Jordan M I,Weiss Y.On spectral clustering:analysis and an algorithm[C]//Advances in Neural Information Processing Systems,2002:849-856.

[62] The bro network security monitor[EB/OL].https://www.bro.org.

[63] Regular expression processor[EB/OL].http://regex.wustl.edu.