软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(1):149−163 [doi: 10.3724/SP.J.1001.2011.03724]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一组提高存储效率的深度包检测算法*

于 强, 霍红卫+

(西安电子科技大学 计算机学院,陕西 西安 710071)

Algorithms Improving the Storage Efficiency of Deep Packet Inspection

YU Qiang, HUO Hong-Wei+

(School of Computer Science and Technology, Xidian University, Xi'an 710071, China)

+ Corresponding author: E-mail: hwhuo@mail.xidian.edu.cn , http://www.cs.xidian.edu.cn

Yu Q, Huo HW. Algorithms improving the storage efficiency of deep packet inspection. *Journal of Software*, 2011,22(1):149−163. http://www.jos.org.cn/1000-9825/3724.htm

Abstract: With the rapid increase in the number of deep packet inspection rules, it is necessary to store deterministic finite automata (DFA) representations of regular expressions efficiently in order to meet the practical requirements of network processing. First, a new hybrid FSM construction method is proposed for compressing the states of DFA. DFAs are built in different ways for the regular expressions. By analyzing the states of the converted DFAs, the distinguished complexities of DFAs become noticeable. This leads to a change in state of the DFA from a quadratic/exponential expression to a linear expression. Next, an efficient compressing algorithm, called Weighted Delayed Input DFA (WD^2FA), is proposed for state transitions of the DFAs. This algorithm can reach a reduction rate of about 95% for the regular expressions with any complexity. The analysis shows that the performance of the WD^2FA is better than the delayed input DFA (D^2FA), and D^2FA is a special case of WD^2FA with weight 0. The experimental results show that the number of states for the FSM can be controlled at the level of linearity, and transitions are reduced to 7% based on the compression states.

Key words: deep packet inspection; regular expression; multi-pattern matching; hybrid FSM; D^2FA (delayed input DFA); WD^2FA (weighted delayed input DFA)

摘 要: 随着深度包检测规则数目的剧烈增长,为了适应网络处理的需求,必须对表示正则表达式的DFA(deterministic finite automata,确定的有限自动机)进行高效的存储.一方面,对DFA的状态点数目进行压缩,提出了一种复合的FSM(有限自动机)的构造方法,通过对正则表达转化成DFA的状态点数目复杂度的分析,将不同复杂度的正则表达式采用不同的方式构建DFA,使得所有平方级和指数级复杂度的状态点数目降低到了线性级.另一方面,对DFA的状态转移数目进行压缩,给出了一种高效的压缩算法,即WD^2FA(weighted delayed input DFA,带权延迟DFA)算法,对于任意复杂度的正则表达式都可以将状态转移数目压缩为原来的5%左右,相对于D^2FA(delayed input DFA,延迟的DFA)有更好的压缩能力,并且使得D^2FA是WD^2FA在权值为0情况下的特例.实验结果表明,有限自动机的状态点数目能够控制在线性级,并且在状态点压缩的基础上将状态转移数目压缩为原来的7%.

关键词: 深度包检测;正则表达式;多模式匹配;复合的FSM;D^2FA(delayed input DFA);WD^2FA(weighted delayed

* 基金项目: 国家自然科学基金(69601003); 青年科学基金(60705004)
收稿时间: 2008-10-17; 定稿时间: 2009-08-19

input DFA)

中图法分类号: TP393 文献标识码: A

由于网络安全的需要,深度包检测变得越来越重要,多模式匹配是最为关键的技术.当前,正则表达式因其灵活和高效的表达能力正在替换传统的精确字符串.表示正则表达式的 DFA(deterministic finite automata)相对于 NFA(nondeterministic finite automata)具有更好的处理性能,但其占用的存储空间是最大的问题.因此,本文主要围绕 DFA 的存储压缩来展开讨论.

1 引 言

深度包检测,即扫描数据包的包头及有效载荷内容,对网络安全及网络监控至关重要.许多重要的网络处理应用都需要深度包检测,比如入侵检测系统、防御系统和防火墙等,且当前大多数系统是软件实现的.

在深度包检测中,多模式匹配是实现数据包扫描的核心技术,即通过扫描一次文本,找出模式集合中的模式在文本中的所有出现.传统的多模式匹配[1,2],其模式是多个精确字符串(例如 apple,banana)的集合.由于通配符、字符集合等正则特征,正则表达式具有很强的表达能力和灵活性,一个正则表达式本身就是一组字符串的集合.正则表达式正在替换精确的字符串成为可选取的模式匹配语言.当前,几个内容检测引擎已经添加了对正则表达式的支持,比如 Snort 入侵检测系统、Bro 入侵检测系统、Linux 应用协议分类器(L7-filter)等.

基于正则表达式的多模式匹配,其原理是将正则表达式构造成 FSM(finite states machine),再用 FSM 对文本进行扫描,从而实现模式在文本中的出现.DFA 因其处理性能比 NFA 好而成为了首选的自动机形式,但 DFA 需要很大的存储空间,这是当前最严重的限制因素.因此,本文主要针对 DFA 的存储压缩进行讨论.

DFA 的存储空间是由状态点数目和每个状态点的状态转移数目共同决定的,文中对这两方面分别进行了探讨.首先,对 FSM 的状态点数目进行压缩,提出了一种复合的 FSM 的构造方法,通过对正则表达式转化成 DFA 的状态点数目复杂度的分析,将不同复杂度的正则表达式采用不同的方式构建 FSM,使得所有平方级和指数级复杂度的状态点数目降低到了线性级.其次,对 DFA 的状态转移数目进行压缩,给出了一种高效的压缩算法,即 WD^2FA(weighted delayed input DFA)算法,该算法尤其对于复杂的正则表达式具有更好的压缩效果,且相对于 D^2FA(dDelayed input DFA)有更强的压缩能力,使得 D^2FA 是 WD^2FA 在权值为 0 情况下的特例.

本文第 2 节介绍 FSM 存储压缩的相关工作.第 3 节针对状态点数目进行压缩.第 4 节是状态转移数目的压缩算法.第 5 节给出压缩算法的整体结构.第 6 节是实验性能结果.最后第 7 节对全文进行总结.

2 背景和相关工作

2.1 自动机的空间和处理性能

为了对存储空间进行压缩,首先考虑正则表达式是如何表示的.正则表达式都是用 FSM 来表示[3],它有两种逻辑形式:DFA 和 NFA.对于任意的正则表达式,都对应有一个具有最少状态点数目的 DFA[3,4].

DFA 用五元组(Q,Σ,q_0,δ,A)表示,其中,Q 为状态点集合,Σ为字符表,q_0 为初始状态点,A 为终止状态点,δ为状态转移函数.一个 DFA 有且只有一个初始状态点作为匹配的起点,之后每扫描文本中的一个字符,则根据转移函数δ确定下一个状态点,当遇到终止状态点(终止状态点可有多个)时,则产生一个模式输出.NFA 与 DFA 的区别在于,NFA 的转移函数δ确定了一个或者多个下一状态点,而 DFA 的转移函数δ确定唯一的下一状态点.

FSM 所需要的存储空间是由状态点数目和每个状态点的状态转移数目共同决定的.在网络处理过程中,字符表是扩展的 ASCII 码,这样,每个状态点有 256 个状态转移.包含数百条正则表达式的规则集,在构建 DFA 时将有数万个状态点,以至于需要数百兆的存储空间.随着正则表达式数目的增加,空间需求将是不可容忍的.

FSM 的空间和处理复杂度见表 1.当输入一个字符时,DFA 能够确定唯一的下一状态点,因此处理复杂度为 $O(1)$.而 NFA 由于它的不确定性,处理一个字符时的最坏复杂度为 $O(n^2)$.DFA 虽然处理复杂度很好,但其实际上

是用更多的状态点记录了 NFA 处理字符的不确定性分枝,以至于空间复杂度要比 NFA 大得多.从理论上讲[3],长度为 n 的正则表达式构建的 DFA 需要状态点数目为 $O(2^n)$,构建的 NFA 需要的状态点数目为 $O(n)$.

对于含有 m 条正则表达式的集合,有 3 种方法可以表示:将其表示成为 m 个独立的 FSM;将其合并成一个 FSM;将其表示成 $k(k<m)$个独立的 FSM,也即将 m 条规则分成 k 组,每组合并成一个独立的 FSM.对于 DFA,多个 DFA 合并可以提高处理性能.若表示成 k 个独立的 FSM,则其处理一个字符的复杂度为 $O(k)$,但 k 值越小,DFA 占用的空间越大.为了达到最佳性能,应该对规则分组采取折衷选择[5].对于 NFA,合并不能提高处理性能.

Table 1 Space and processing complexity of FSM

表 1 FSM 的空间和处理复杂度

	One Regex of length n		m Regex not compiled together		m Regex compiled together	
	Storage cost	Processing complexity	Storage cost	Processing complexity	Storage cost	Processing complexity
NFA	$O(n)$	$O(n^2)$	$O(nm)$	$O(n^2m)$	$O(nm)$	$O(n^2m)$
DFA	$O(2^n)$	$O(1)$	$O(2^n m)$	$O(n)$	$O(2^{nm})$	$O(1)$

Note: Storage cost is the number of states; Processing complexity is the complexity of processing a single character.

第 2.2 节和第 2.3 节介绍当前对 DFA 的优化算法和相关工作,包括处理性能(或者称为吞吐量)的优化和存储空间的优化,此外,还有硬件上的相关应对方案.文献[6]较全面地介绍了深度包检测问题和相关算法.

2.2 提高吞吐量的算法

吞吐量(throughout)是指 FSM 经过一次状态转移能够处理的字符个数,用以衡量 FSM 的处理性能.DFA 的处理能力为经过一次状态转移处理一个字符.一种改进方法是,对 DFA 的状态转移表进行扩展[7],即对字符表进行笛卡尔乘积,生成新的状态转移表,使其在处理时每次能够读入多个字符.状态转移扩展的原理如下:

$$\delta(u,ab)=\delta(\delta(u,a),b).$$

在这个式子中,u 代表初始状态点,a 与 b 代表字符激励.利用扩展的状态转移表,状态点 u 经过两个字符 ab 的激励可以直接产生输出,输出的下一状态点等于状态点 u 先经过字符激励 a 再经过字符激励 b 产生的下一状态点.

对转移表的扩展,使得状态转移数目剧增,也即存储空间变得很大,因此必须对其进行压缩.

2.3 存储压缩算法

正则表达式的 rewrite 技术[5],其前提是采用最左最短匹配方式(left shortest matching)扫描数据,在不影响匹配结果的情况下对部分正则表达式进行改写,从而减少了 DFA 的状态点数目.

表压缩算法[7,8]是将字符表中的字符进行等价划分和重新编码(当同一等价类中的字符,作为任意状态点的激励时,产生相同的下一状态点),字符个数从 256 减少到了等价类的个数,从而降低了状态转移的数目.

D^2FA 存储压缩技术[13]是针对状态转移的压缩技术.在 DFA 中,对于任意两个状态点 u 和 v,某些相同的激励可能产生相同的状态点输出,即 $\delta(u,a)=\delta(v,a)$.对于所有类似字符 a 产生的状态转移,只在状态点 u 中加以保留,而在状态点 v 中去除所有的这些状态转移并为其添加一个称为默认状态转移(default transitions)的状态转移到状态点 u.这样,当在状态点 v 中匹配类似 a 的字符时,首先默认转移到 u,再由 u 决定下一状态点.

位图压缩[8,10]也是一种针对状态转移的压缩算法,它用 256 个连续的 bit 位对应于 ASCII 码,标识出字符激励的类别.文献[10]中,从某个状态点出发,若存在某个字符为激励,则将对应的 bit 位标识为 1,否则,标识为 0.文献[8]中,从某个状态点出发,若某个字符激励要经由 D^2FA 的默认转移,则将对应的 bit 位标识为 1,否则,标识为 0.

规则集的分组(grouping)技术[5]将多个 DFA 合并成一个 DFA,以提高处理性能.为了不增加额外的存储空间,采用如下处理方法:对所有 DFA 进行划分,划分子块的数量要达到最少,并且每一个划分子块合并成的 DFA 状态点数目不超过对应划分子块各 DFA 状态点数目的总和.

此外,采用硬件平台,借助并行技术可以达到很好的性能.硬件结构有两种实现策略:其一,将 DFA 制成硬件逻辑[7,11]:当处理一个字符时是并行处理的,使得处理字符的复杂度为 $O(1)$.采用这种策略要考虑规则集的更新,

即硬件逻辑需要重新配置(reconfiguration),于是 FPGA 是一个很好的选择[12].另外,当规则集数目较大时,硬件资源消耗也大,必须进行存储压缩;其二,基于存储器的硬件设计[5,9,13]:将 FSM 存储在存储器中,这样,可以根据存储器的数目和类型进行灵活的设计.但为了将 FSM 存于片内存储器中,对空间的压缩有更为严格的要求.

3 状态点的压缩

正则表达式的一些正则特征导致了对应的 DFA 要占用巨大的存储空间.文献[5]在分析了正则特征的基础上提出了正则表达式的 rewrite 技术,从而减少了一部分 DFA 的状态点数目.本节对正则特征进行了更加详细的分析,之后提出复合的 FSM 的解决方案,从而使所有平方级和指数级复杂度的状态点数目降低到线性级.第 3.1 节对正则表达式生成 DFA 的状态点数目的复杂度进行了分析.在此基础上,第 3.2 节提出了复合的 FSM 的构建方法.第 3.3 节~第 3.5 节给出了实现复合的 FSM 的相关技术.

3.1 构建复杂度分析

构建复杂度在这里是指正则表达式生成 DFA 的状态点数目的复杂度.长度为 n 的正则表达式最坏情况下的构建复杂度为 $O(2^n)$[3].然而在实际情况中,构建复杂度大多数呈线性级或者平方级.经过对具体实验数据的分析,表 2 给出了详细的构建复杂度,其中 K 代表正则表达式的长度.

Table 2 Complexity of constructing DFA for regular expression

表 2 构建正则表达式的 DFA 的复杂度

Complexity	Regular expressions	# of states
Linear size	^*ABCD*	*K*+1
	^*AB*.**CD*	*K*+1
	.**ABCD*	*K*+1
	.**AB*.**CD*	*K*+1
	^*AB*.{*j*}*CD*	*K*+1+*j*
	^*AB*.{*j*+}*CD*	*K*+1+*j*
	^*AB*.{0,*j*}*CD*	*K*+2*j*+1
Quadratic size	^*A*+.{*j*}*D*	*K*+(*j*+1)(*j*+2)/2
Exponential size	^*AB*[^\n]**CD*.{*j*}*EF*	$O(K+2^j)$
	.**AB*.{*j*}*CD*	$O(K+2^j)$
	.**A*[*A*-*Z*]{*j*+}*D*	$O(K+2^j)$

我们以^*AB*.{0,*j*}*CD* 为例,说明 DFA 的线性构造过程.在构建 DFA 时,*j* 每增 1,则需要增加一个并列结构的分枝,引出一个 *C* 与非 *C* 的状态点,这样的状态点数目为 *K*+2*j*+1.需要平方复杂度构造 DFA 的正则表达式有^*A*+.{*j*}*D*,其状态点数目提升的原因是 DFA 需要记录 *A*+与.{*j*}的重叠.在构建 DFA 时,状态点数目的多项式级增长是由字符集合和长度限制{*j*}的组合引起的,与 *K* 无关.如果正则表达以.*开头,并包含字符集合与长度限制的组合,构建时状态点数目的增长就呈现出了指数级的复杂度,当扫描文本到达某个状态点时,若出现不能继续向前匹配的情况,就需要从当前状态点回退,而状态点数目的指数级增长原因就在于要记录大量的回退信息.^*AB*[^*n*]**CD*.{*j*}*EF* 是一种较为特殊的形式,其虽然以定位符^开始,但中间的[^*n*]*充当了.*的作用,使得整个正则表达式与.**CD*.{*j*}EF 一样具有指数级的复杂度.

3.2 复合的FSM

根据正则表达式构建 DFA 状态点数目的复杂度,将不同复杂度类型的正则表达式构建成不同类型的 FSM,由这些不同类型的 FSM 组合成的 FSM 称为复合的 FSM(hybrid FSM).

高复杂度正则表达式中的几个或者一个所占用的存储空间就是无法容忍的.因此,复合的 FSM 的实现目标为:从规则集中把所有构建成 DFA 后对应有平方级和指数级状态点数目的正则表达式无一遗漏地挑选出来,并转换成特殊的 DFA,以将状态点数目的复杂度降低到线性级.

复合 FSM 的实现方法如图 1 所示.先对正则表达式集合(规则集)按照构建 DFA 的复杂度进行分类.然后,线性级复杂度对应的正则表达式转化为普通的 DFA;平方级复杂度对应的正则表达式,具有^*A*+.{*j*}形式的先用

rewrite 技术[5]进行改写,再转化为普通的 DFA,具有^*A*+.{*j*}*D* 形式的,转化为带计数器的 DFA;指数级复杂度对应的正则表达式,具有.**AB*.{*j*}和^*AB*[^*n*]**CD*.{*j*}形式的用 rewrite 技术转化为普通的 DFA,具有.**AB*.{*j*}*CD* 和^*AB*[^*n*]**CD*.{*j*}*EF* 形式的,转化为 Lazy DFA[14−17].

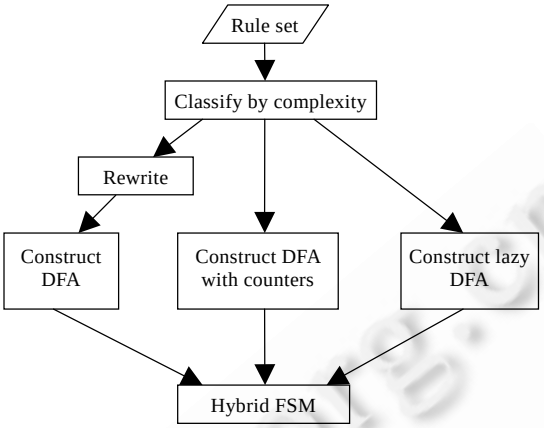


Fig.1 Constructing hybrid FSM

图 1 构建复合的 FSM

3.3 对Rewrite技术的分析和扩展

Rewrite 技术[5]的原理,是在不影响匹配结果的情况下(采用最左最短的匹配方式),对具有某些特征的正则表达式进行改写,从而达到减少 DFA 状态点数目的目的.Rewrite 技术形成了两条改写规则:

规则 1. 将具有^*A*+.{*j*}特征的正则表达式改写为^*A*.{*j*},这样,状态点数目降低到了 *K*+*j*.

规则 2. 主要针对.**AB*.{*j*}或者.**AB*[*A*-*Z*]{*j*}形式的正则表达式,实际上就是遵从最左最短的方式进行匹配,这样生成的 DFA 状态点数目为 *K*+*j*.

基于第 3.1 节对正则表达式复杂度的分析,这里增加一条 rewrite 规则,即:

规则 3. 对^*AB*[^*n*]**CD*.{*j*}形式的正则表达式,用最左最短匹配方式构建 DFA,这样,状态点数目为 *K*+*j*.

Rewrite 技术可以将一部分平方级和指数级构建复杂度的正则表达式改写为具有线性级复杂度的正则表达式,但只能改写以长度限制(即{*j*})为后缀的正则表达式,无法处理^*A*+.{*j*}*D*,^*AB*[^*n*]**CD*.{*j*}*EF* 或者.**AB*.{*j*}*D* 形式的正则表达式.也即这 3 种形式的正则表达式一旦按照前边的方法改写,将无法得出正确的匹配输出.

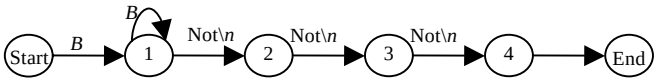
针对 rewrite 技术的不足,本文采用如下应对方法:对于^*A*+.{*j*}*D* 形式的正则表达式,提出了带计算器的 DFA;对于.**AB*.{*j*}*D* 和^*AB*[^*n*]**CD*.{*j*}形式的正则表达式,采用 Lazy DFA.第 3.4 节和第 3.5 节描述了这些方法.

3.4 带计数器的DFA

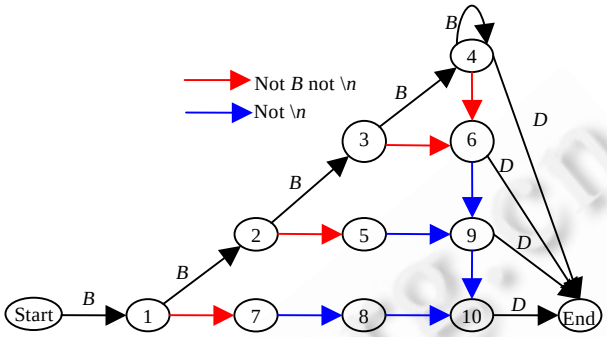
3.4.1 结构分析

以^*B*+[^*n*]{3}D 为例,分析一下具有平方级构造复杂度的正则表达式转化成 FSM 的结构特点.图 2(a)为^*B*+[^*n*]{3}*D* 对应的 NFA,其状态点数目为 *O*(*n*).在最坏情况下,处理复杂度为 $O(n^2)$.假设输入串为 BBBBD,匹配过程中形成的状态集依次为{0},{1},{1,2},{1,2,3},{1,2,3,4},{1,2,3,4,5},由于最后一个状态集中含有终止状态 5 而使得匹配终止.将各个状态集中的状态点数目加起来,可以得到 NFA 处理时的转移次数.这样,^*B*+[^*n*]{*j*}*D* 对应的 NFA 在最坏情况下的转移次数为(*j*+1)(*j*+2)/2.图 2(b)为^*B*+[^*n*]{3}*D* 对应的 DFA,由于字符 *B* 在 *B*+与[^*n*]{3}中重叠出现,DFA 必须记录输入字符 *B* 的每种可能路径,这样使其具有了平方级的状态点数目.精确地讲,描述 *B*+[^*n*]{*j*}的 DFA 所需要的状态点数目为(*j*+1)(*j*+2)/2,如图 2(b)中的状态点 1~状态点 10.

通过上述分析可知,DFA 中记录路径需要的状态点数目与 NFA 在最坏情况下需要的处理次数相等.这并非巧合,而是反映了用子集法构造 DFA 的原理,同时也展现了平方级复杂度的 DFA 结构是如何形成的.



(a) NFA for Regex ^*B*+[^\n]{3}*D*
(a) ^*B*+[^\n]{3}*D* 对应的 NFA

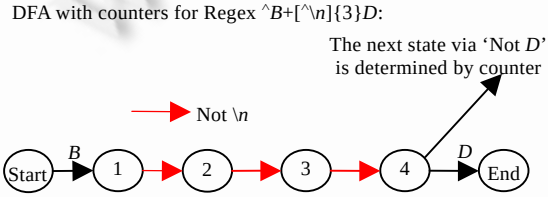


(b) DNA for Regex ^*B*+[^\n]{3}*D*[5]
(b) ^*B*+[^\n]{3}*D* 对应的 DFA[5]

Fig.2 FSM for Regex ^*B*+[^\n]{3}*D*
图 2 ^*B*+[^\n]{3}*D* 对应的有限自动机

3.4.2 构建带计数器的 DFA

我们以^*B*+[^\n]{3}*D* 为例,解释具有计数器的 DFA 的构造方法.如图 3(a)所示:首先将正则表达式改写为^*B*[^\n]{3}*D*,并构造相应的 DFA;之后引入一个计数器 *counter*,用以记录历史路径,即记录从状态点 1 开始到达状态点 4 终止的连续的 *B* 的个数.这样,状态点 4 的转移是由当前状态点及计数器共同决定的.状态点 4 的转移信息如图 3(a)所示,计数器的取值为 0~3,相应的状态 4 有 4 种状态转移信息.



The transmission operation of state 4:
counter==0: *move*(4,*D*)=5
counter==1: *move*(4,*D*)=5;
move(4,not \n & *D*)=4, *counter*=*counter*−1
counter==2: *move*(4,*D*)=5;
move(4,not \n & *D*)=4, *counter*=*counter*−1
counter==3: *move*(4,*D*)=5;
move(4,*B*)=4;
move(4,not *B* & *D* & \n)=4, *counter*=*counter*−1

(a) Example for DFA with counters
(a) 带计数器 DFA 的示例

```
SCAN_DFA_With_COUNTER(Text)
1   state=the start state, counter=0
2   while get char from Text
3     if state is not internal state
4       state=move(state,char)
5     if state is internal state and is not internal endState
6       state=move(state,char)
7       if both char this time and last time are 'B'
8           counter++
9     if state is internal endState
10      switch counter
11        case  0:      state=move(state,char)
12        case  1 to j–1: if char is 'D'
13                          state=move(state,char)
14                        if char is neither 'D' nor '\n'
15                          state doesn't change, counter––
16        case  j:      if char is 'D'
17                          state=move(state,char)
18                        if char is 'B'
19                          both state & counter don't change
20                        if char is not 'D', not 'B' and not '\n'
21                          state doesn't change, counter––
```

(b) Scanning algorithm for DFA with counters
(b) 带计数器 DFA 的扫描算法

Fig.3 DFA with counters
图 3 带计数器的 DFA

本文将^*B*[^\n]{*j*}*D* 中表示[^\n]{*j*}的第 1 个状态点(图 3(a)中的状态点 1)和最后一个状态点(图 3(a)中的状态点 4)分别称为内部起始状态点(internal startState)和内部终止状态点(internal endState);将内部起始和终止状态点以及它们之间的状态点称为内部状态点(internal state).带计数器的 DFA 构建方法为:先将^*B*+[^\n]{*j*}*D* 改写成^*B*[^\n]{*j*}*D* 再构建成标准的 DFA,同时记录内部状态点的标号.在扫描时要做一些与计数器相关的处理:计数器初始化为 0,当到达内部起始状态点后,触发计数器并记录从内部起始状态点出发的连续 *B* 的个数,每有一个连续的 *B*,则计数器加 1;到达内部终止状态点后,要结合计数器的值共同决定下一转移状态.^*B*+[^\n]{*j*}*D* 对应的计数器具有 0~*j* 的取值范围,因而内部终止状态具有 *j*+1 种状态转移信息.通过图 3(a)可以发现,实际上,状态转移只有 3 种,即 *counter* 的值划分为 3 类:{0},{1~*j*–1}和{*j*}.带计数器 DFA 的扫描处理算法如图 3(b)所示.

用上述方法构造具有计数器的 DFA 实现了对^*B*+[^\n]{*j*}*D* 形式的正则表达式的改写,使状态点数目从平方级降低到了线性级.更为精确地,将表示[^\n]{*j*}这一部分的需要的状态点数目从(*j*+1)(*j*+2)/2 减少到了 *j*+1.

3.5 Lazy DFA

对于 rewrite 技术无法改写的形如.**AB*.{*j*}*D* 和^*AB*[^\n]**CD*.{*j*}*EF* 的正则表达式,其具有指数级的状态点数目,这里将其构建成 lazy DFA.它介于 NFA 和 DFA 之间,在 NFA 扫描处理时计算并存储实际需要的一些 DFA 状态点,状态数目的复杂度为 *O*(*n*),从而将对应的正则表达式的构建复杂度从指数级降低到了线性级.

4 状态转移的压缩

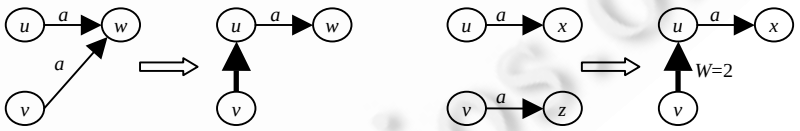
为了对 DFA 状态转移的数目进行压缩,我们给出了一种高效存储的算法,即 WD^{2}FA(weighted delayed input DFA,带权延迟 DFA).WD^{2}FA 比 D^{2}FA 具有更好的压缩性能,且 D^{2}FA[9]是 WD^{2}FA 权值为 0 情况下的特例.

4.1 WD^{2}FA

4.1.1 构造思路

对两种 DFA 状态而言,除了有可能存在大量的相同字符激励指向相同的下一状态之外(D^{2}FA 的构造思路,如图 4(a)所示),如下情况也是经常存在的:如图 4(b)所示,取两个状态点 *u* 和 *v*,对于大量相同的字符激励 *a*,使得 $\delta(v,a)=\delta(u,a)+W$,其中,*W* 为一个整数.类似于 D^{2}FA 的构造思路,只在状态 *u* 中保留这些相同字符激励的状态转移信息,*v* 中去除这些字符激励对应的转移信息并添加一个带权值(*W*)的默认状态转移到状态 *u*.图 4(b)中,加粗的箭头代表默认转移.假设状态点 *z* 的编号值比状态点 *x* 大 2,则权值 *W* 设置为 2.当在状态点 *v* 匹配 *a* 这样的字符时,先由默认转移转换到状态点 *u*,则下一状态的值为权值 2 加上 *u* 指向的下一状态点 *x*,即得到状态点 *z*.

在图 4(b)中,只有当状态点 *x* 和 *z* 是同一个状态点时(如图 4(a)所示),才能用 D^{2}FA 算法进行压缩.因此我们可以发现,D^{2}FA 是 WD^{2}FA 的所有默认转移权值都为 0 情况下的特例.



(a) Constructing strategy for D^{2}FA
(a) D^{2}FA 的构造思路

(b) Constructing strategy for WD^{2}FA
(b) WD^{2}FA 的构造思路

Fig.4 Constructing strategy for WD^{2}FA

图 4 WD^{2}FA 的构造思路

4.1.2 一个例子

我们以正则表达式^*B*+.{3}*D* 为例进一步说明 WD^{2}FA 的工作过程,其状态转移矩阵见表 3.在表中,‘–’表示没有下一转移状态,在实际中应该回到起始状态点.为了更清楚地说明问题,其值取为–1.DFA 的每个状态点有 256 个状态转移,这样,整个 DFA 有 3 072 个状态转移.经 WD^{2}FA 压缩后,一共只需记录 274 个状态转移.

Table 3 WD²FA corresponding to ^*B*+.{3}*D*

表 3 ^*B*+.{3}*D* 对应的 WD²FA

State	Stimulate			
	B	*D*	Not *B* Not *D* (254 characters)	# of transitions
0	1	—	—	256
1	3	Default to 0, *weight*=3	Default to 0, *weight*=3	2
2	4	Default to 0, *weight*=5	Default to 0, *weight*=5	2
3	Default to 1, *weight*=3	Default to 1, *weight*=3	Default to 1, *weight*=3	1
4	Default to 2, *weight*=3	Default to 2, *weight*=3	Default to 2, *weight*=3	1
5	Default to 2, *weight*=4	Default to 2, *weight*=4	Default to 2, *weight*=4	1
6	Default to 1, *weight*=7	Default to 1, *weight*=7	Default to 1, *weight*=7	1
7	Default to 2, *weight*=-5	11	Default to 2, *weight*=-5	2
8	Default to 2, *weight*=3	11	Default to 2, *weight*=3	2
9	Default to 2, *weight*=4	11	Default to 2, *weight*=4	2
10	Default to 1, *weight*=7	11	Default to 1, *weight*=7	2
11	—	Default to 0, *weight*=0	Default to 0, *weight*=0	2
				Total: 274

4.1.3 扫描处理

在用 WD²FA 处理字符时,从某个状态点出发,先检测输入字符对应的状态转移信息是否为默认转移:如果是非默认转移,则根据状态转移信息直接到达下一状态点;否则,根据默认转移到的状态点和默认转移权值共同决定下一状态点,然后再重复上述操作.若经由多次默认转移,则要对每次获取的默认转移权值进行累加.例如,取表 3 中的状态点 3,输入字符为 *A*,默认转移到 1,权值为 3;然后从状态点 1 出发,经字符 *A* 默认转移到 0,权值为 3,累加为 6;再从状态点 0 出发,到达下一状态点为–1,没有默认转移,加上权值 6,则得到最终的状态点为 5.

4.2 形式化描述

在 DFA 定义的基础上,第 4.2 节定义了一些与 WD²FA 相关的概念,以便于 WD²FA 的分析和算法设计.

给定一个 $DFA=\{\Sigma,\delta,Q,q_0,A\}$,$\Sigma$为字符表,$\delta$为转移函数,$Q$ 为状态点集合,q_0 为初始状态,A 为接受状态.

定义 1(相似). 对于任意 $u,v\in Q,u\neq v$,任取 $a,b\in\Sigma$,如果存在常数 W,使得 $\delta(u,a)=\delta(v,a)+W$,且 $\delta(u,b)=\delta(v,b)+W$,则称激励字符 *a* 与 *b* 在状态点 *u* 和 *v* 中是相似的.

假设字符表为{*a*,*b*,*c*,*d*,*e*,*f*},状态转移信息见表 4.字符 *a*,*b*,*e* 是相似的,即任取字符 $x\in\{a,b,e\}$,有 $\delta(u,x)=\delta(v,x)+1$.同理,字符 *c*,*d* 是相似的.特别地,根据定义,一个单独的字符与其自身是相似的,例如字符 *f*.

Table 4 Transitions of the states *u* and *v*

表 4 状态 *u* 和 *v* 的转移信息

State	Stimulate					
	a	*b*	*c*	*d*	*e*	*f*
u	2	2	3	4	6	10
v	1	1	5	6	5	7

定义 2(相似划分). 对于任意 $u,v\in Q,u\neq v$,存在字符表 Σ 的子集 $\Sigma_1,\Sigma_2,\ldots,\Sigma_n$ 且 $\bigcup\Sigma_i=\Sigma,\forall i,j\in[1,n],i\neq j$ 有 $\Sigma_i\cap\Sigma_j=\varnothing$,使得 $\forall a\in\Sigma_i$ 有 $\delta(u,a)=\delta(v,a)+W_i$ 且 $\forall i,j\in[1,n],i\neq j$ 有 $W_i\neq W_j$,则称 $\Sigma_1,\Sigma_2,\ldots,\Sigma_n$ 是 Σ 在状态点 *u* 和 *v* 中的相似划分.

$\Sigma=\{a,b,c,d,e,f\}$ 在状态点 *u* 和 *v* 中的相似划分为 $\Sigma_1=\{a,b,e\},\Sigma_2=\{c,d\},\Sigma_3=\{f\}$,且 $W_1=1,W_2=-2,W_3=3$.

定义 3(相似度). 对于任意 $u,v\in Q,u\neq v$,设 Σ 的相似划分为 $\Sigma_1,\Sigma_2,\ldots,\Sigma_n$,令

$$D=\max_{1\leqslant i\leqslant n}|\Sigma_i| \tag{1}$$

称其为状态点 *u* 和 *v* 的相似度.

例子中,状态点 *u* 和 *v* 的相似度 *D*=3,因为划分子块 $\Sigma_1=\{a,b,e\}$ 具有最多的字符数目.

划分子集的个数可能只有 1 个或 $|\Sigma|$ 个,相应的 *D* 取值为 $|\Sigma|$ 或 1.这样,*D* 的取值范围为 $[1,|\Sigma|]$.后文描述时用 D_{uv} 或者 D_{vu} 来表示状态点 *u* 和 *v* 的相似度,D_{uv} 与 D_{vu} 是相等的,它们表示同一个信息,即相似度 *D*.

定义 4(默认转移和默认转移权值). 对于任意 $u,v\in Q,u\neq v$,设 Σ 的相似划分为 $\Sigma_1,\Sigma_2,\ldots,\Sigma_n$,状态点 *u* 和 *v* 的相似

度 D 对应的划分子块为Σ_i,任取 $a\in\Sigma_i$,有$\delta(u,a)=\delta(v,a)+W_i$.从状态点 u 的状态转移中去除Σ_i 中字符对应的状态转移,同时给状态点 u 添加一个从其出发到状态点 v 的转移,即Σ_i 中字符对应这个新添加的状态转移,称这个新添加的状态转移是从状态点 u 到状态点 v 的默认转移(default transitions).此外,称 W_i 为状态 u 到状态 v 的默认转移权值,简称为状态 u 到状态 v 的权值,记作 W_{uv}.

延用上边的例子,状态点 u 和 v 的相似度 D 对应的划分子块为$\{a,b,e\}$,因此,在状态 u 中去除字符 a,b,e 对应的状态转移信息,然后添加一个到状态 v 的默认转移,状态 u 到状态 v 的默认转移权值为 W_{uv}=1.

性质 1. 对于任意 $u,v\in Q,u\neq v$,有 $W_{uv}=-W_{vu}$.

证明:从状态 u 到状态 v,在划分子块Σ_i 中,$\forall a\in\Sigma_i$,有$\delta(u,a)=\delta(v,a)+W_i$,即 $W_{uv}=W_i$.改写一下表达式有$\delta(v,a)=\delta(u,a)+(-W_i)$,实际上就是状态 v 到状态 u 的表达式,即 $W_{vu}=-W_i$.因此,$W_{uv}=-W_{vu}$. □

4.3 算法描述

4.3.1 问题分析

本节通过对 3 个问题的回答进行展开,并引出 WD^{2}FA 的数据结构,最后给出 WD^{2}FA 的实现目标.

问题 1. 对于状态集中的每个状态点是不是都要建立默认转移关系?

在这里,给某个状态点建立默认转移关系是指,要么从这个状态点出发有指向其他状态点的默认转移,要么有默认转移到达这个状态点,或者两者兼具.正则表达式一般由几个至几十个有效字符组成,例如 Snort 系统中的 NNTP 规则^*SEARCH*\s+[^\n]{1024},起主要作用的字符为{*S,E,A,R,C,H,*\s,\n},只有 8 个.而字符表有 256 个字符,在构建 DFA 时,从每个状态点出发还要为不起主要作用的 200 多个字符建立转移信息,而这些字符往往都指向同一个下一状态.换言之,任取状态集中的两个状态点 u 和 v,一般情况下,它们的相似度 D_{uv} 大于 200.

为了争取最大化的空间存储压缩,我们为状态集中的每一个状态点都建立默认转移关系,但有一定的限制条件,建立默认转移关系后的状态机是有向图,必须保证默认转移不能形成环,否则处理字符时将陷入死循环.因此,至少有一个状态点只有指向它的默认转移而没有从它出发指向其他状态的默认转移.

问题 2. 如果从某个状态点出发有默认转移,那么允许它有几个默认转移?

结合 WD^{2}FA 处理字符的原理,答案是只允许有一个到其他状态点的默认转移,否则将产生多分枝现象.

由上可以引出表示默认转移的数据结构,默认转移边构成了有向图.问题 1 表明,在此有向图中不存在环;问题 2 表明,从某个状态点出发最多只有一个有向边.所以,此有向图是树或者森林.若一个状态点没有从它出发指向其他状态点的默认转移,那么这个状态点是根结点.若这样的状态点只有一个,则此有向图是树,否则是森林.

问题 3. 处理一个字符时可能要经由多次默认转移,那么默认转移的次数要不要限制?

定义 5(默认转移长度). 在由默认转移构成的树中,树的最大深度称为默认转移长度,记作 L.

在用 WD^{2}FA 处理字符时,处理一个字符的复杂度为 $O(L)$.如果对 L 的值不加以限制,它的值可能较大,这样虽然压缩强度大,但却影响了处理性能.因此,处理一个字符时,默认转移长度 L 要加以限制.

限制默认转移长度 L,就是限制树的深度.这里只构建一棵树,在原有的树中不断地添加状态点 u 到树中现有的状态点 v,使得任取树中现有的状态 w,有 $D_{uv}=\max\{D_{uw}\}$.若存在多个满足条件的 v,则取现有树中的深度最小的状态点.如果添加的树枝使树深超过了 L,那么重新寻找一个原树中的结点,直至满足默认转移的长度限制为止.

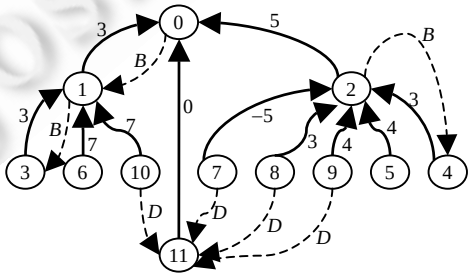


Fig.5 Default transition tree for Regex ^*B*+.{3}*D*

图 5 ^*B*+.{3}*D* 的默认转移树

通过对 3 个问题的分析,可以得出构造 WD^{2}FA 的要点.在构建 WD^{2}FA 时,为状态集中的所有状态点都建立默认转移关系,且由默认转移形成的有向图是树,并且要对默认转移长度(树深)加以限制.图 5 给出了一个默认转移树的示例,其中加粗的实线边代表默认转移,边上的数字是默认转移的权值,虚线边代表非默认的状态转移,边上的字母是字符激励,从而形成了一棵以状态

点 0 为根结点、最大树深(长度限制 L)为 2 的树 2.

4.3.2 算法描述

如图 6 所示,我们首先给出了表示两个状态点相似度的数据结构以及求解两个状态点间相似度的算法.

```
Data Struct:
struct resemble
{
    int   resembleDegree;
    int   weight;
    bitset 〈256〉 resembleBlock;
};
The variables in the algorithm:
w_maxPB: default transition weight
          corresponding to the dividing block
          which has the most characters currently.
w: temp variable for counting default
   transition weight.
PB[w]: dividing block with weight w
```

```
GET_RESEMBLEDEGREE(u,v)
1   w_maxPB=0
2   for each char in Σ do
3     w=δ(u,char)−δ(v,char);
4     insert char to PB[w]
5     if the number of characters in PB[w] is large than w_maxPB
6       w_maxPB=w
7   resemble rsb
8   rsb.resembleDegree=the number of characters in PB[w_maxPB]
9   rsb.weight=w_maxPB
10  set bits of rsb.resembleBlock according to PB[w_maxPB]
11  return rsb
```

Fig.6 Data structure and algorithm for resemble degree computing

图 6 相似度的数据结构和求解算法

在数据结构中,相似划分块(*resembleBlock*)是一个 256 个 bit 的位图,当字符对应的状态转移是默认转移时,则相应的 bit 位置为 1,否则,置为 0.通过这种方法,在匹配处理中输入一个字符激励后,我们就能够判别对应的下一状态转移是否为默认转移.从算法中可以得出计算两个状态点间相似度的时间复杂度为 $O(|\Sigma|)$.

为了表示整个状态机的默认转移关系,要构建一棵树,树枝是两个结点间的默认转移.考虑这样一个无向完全图:顶点集合为状态机的状态点集合 Q,任取两个顶点 u 和 v,u 和 v 之间存在一条无向边,边的权值为顶点 u 和 v 的相似度 D_{uv}.在为所有状态点构建默认转移时,若不考虑长度限制,那么就是从这个无向完全图中获取一个最大权值生成树.由于默认转移长度要加以限制,因此我们要构建树深不超过长度限制的最大权值生成树.

构建生成树时利用 prim 算法,并在其基础之上限制树的深度不超过长度限制的值,WD^{2}FA 的带长度限制的默认转移生成树算法如图 7 所示.建立好生成树之后,所有的状态点都包含在了树中,因此只需从树根开始遍历所有结点,并在非根结点中添加到其父结点的默认转移关系,则为整个 FSM 建立了默认转移关系.

图 8 给出了 WD^{2}FA 扫描处理算法的伪代码,处理单个字符的复杂度为 $O(L)$.其中,L 为默认转移的长度限制.

```
INPUT: Undirected complete graph 〈Q,D〉, Q is the set of vertices (state set of DFA), D is the set of edges, the value of each edge
       is resembleDegree between two vertices. Length restriction of default transitions: L;
OUTPUT: MaxSpanTree whose depth is not large than L.
MAXSPANTREE
1.  add q0 to the empty vertex set Q1
2.  Q2=Q−Q1
3.  initialize the root of spanTree with q0
4.  while Q2 is not empty do
5.      for all vertices in Q1 and Q2 do   find u∈Q2 and v∈Q1, such that Duv is the max.
6.      delete u from Q2 and add u to Q1
7.      if depth of spanTree is large than L when linking u to v
8.          choose a new vertex x in Q1, such that Dux is max.
9.          link u to x in spanTree
10      else
11          link u to v in spanTree
12      redress the root of spanTree, such that the depth of spanTree is min.
```

Fig.7 Algorithm for max*SpanTree* with length restrictions

图 7 带长度限制的默认转移生成树算法

```
SCAN_WD²FA(Text)
1  state=the start state
2  while get char from Text do
3     w=0
4     while isDefaultTranslation(state,char) do
5        state=nextState_default
6        w+=weight
7     state=move(state,char)+w
```

Fig.8 Processing algorithm for WD²FA

图 8 WD²FA 的处理算法

5 存储算法的整体结构及分析

结合前边的内容,本节给出了一个高效存储的深度包检测算法的实现结构,如图 9 所示.

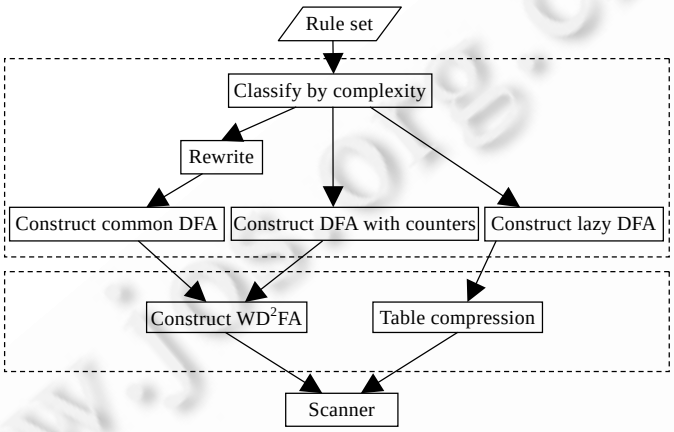


Fig.9 Overall structure of compressing algorithm

图 9 压缩算法的整体结构

首先对 FSM 的状态点数目进行压缩,如图中上方的虚线框内所示.其次,在状态点数目压缩的基础上,对 FSM 的状态转移数目进行压缩,如图中下方虚线框内所示.

采用图中的算法框架,可以对 FSM 占用的存储空间实现最大化的压缩.通过状态点数目的压缩,将规则集进行分类,然后分别构建不同的FSM,从而形成复合的FSM,使得所有平方级和指数级复杂度的数目降低到线性级.进一步对它们的状态转移数目进行压缩,我们可以将复合的 FSM 内的自动机看作两类:一类是 DFA,对应图中普通的 DFA 和带计数器的 DFA;另一类是 NFA,对应图中的 Lazy DFA.对 DFA 采用构建 WD²FA 的方式进行压缩,通过实验,其结果表明,可以将状态转移数目稳定地压缩为原来的 5%左右.对 NFA 采用表压缩技术进行压缩,经实验统计,表压缩可以将字符表中 256 个字符压缩为 20 多个.这样,表压缩可以将转移数目压缩为原来的 10%左右.对于这些算法的结合运用,在第 6 节中给出了详细的实验数据说明.

6 实验数据比较和分析

6.1 状态点数目的压缩结果

实验使用了 Snort 系统 2007 年 3 月注册版的规则集,从中提取了 2 087 条正则表达式.表 5 给出了要构建的不同的 FSM 对应正则表达式的数目、平均长度、平均长度限制(正则表达式{}内的数值)和构建 DFA 占用的空间等信息.复合的 FSM 对表中的 5 个类别分而治之,改进后的空间性能见表 6.压缩前后,对 DFA 占用空间的统计都没有使用状态转移的压缩.这样,复合的 FSM 占用的总空间大约为 512M.

Table 5 Regular expressions classes and RAM sizes

表 5 正则表达式分类及空间占用

Category	Amount	Proportion to total amount (%)	Average length	Average length restrictions	Constructing complexity	Storage cost of DFA (bytes)
Common DFA	1 915	91.76	118	28	Linear	59 269 442
Rewrite rule 1	11	0.53	25	360	Quadratic	653 015 554
DFA with counters	5	0.24	63	211	Quadratic	58 918 912
Rewrite rule 2	81	3.88	43	520	Exponential	To infinite
Lazy DFA	75	3.59	212	788	Exponential	To infinite

Table 6 Comparisons of space occupation before and after compression

表 6 压缩前后空间占用情况

Category of FSM	Original storage cost (bytes)	Storage cost after compression (bytes)	Compression rate (%)
Common DFA	59 269 442	59 269 442	0
Rewrite rule 1	653 015 554	2 070 018	99.68
DFA with counters	58 918 912	1 411 074	97.61
Rewrite rule 2	To infinite	17 912 449	>99
Lazy DFA	To infinite	430 966 882	>99
Total storage cost of hybrid FSM: 511 629 865 bytes			

关于处理性能,rewrite 技术对其并无影响.对于带计数器的 DFA,当触发计数器时,与标准的 DFA 相比,所需要的处理仅仅是对计数器进行赋值操作以及最后一个内部状态点对计数器的判断处理操作,将计数器的值存储在寄存器中,这样具有与标准的 DFA 相当的处理性能.对于 Lazy DFA,开始构建和存储时是 NFA,然后在扫描处理时计算并存储第 1 次用到的 DFA 状态点,当下次使用这些状态点时,只需查表即可.Lazy DFA 实现时要考虑如何减小其查表时带来的额外开销,可以用二分查找或哈希查找代替线性查找.

6.2 状态转移数目的压缩结果

本节通过比较压缩前后的状态转移数目来验证 WD^2FA 的压缩性能.在 Snort 规则集中有一小部分指数级构造复杂度的正则表达式无法直接构造成 DFA,相对而言,Linux L7-filter 规则集要简单一些.因此,实验选取了 Linux L7-filter 的规则集,并整理出 113 条正则表达式.实验结果见表 7,压缩比是指用压缩掉的数目除以原来 DFA 的总数目.对于所有的 113 条规则,WD^2FA 的压缩比是 94.26%,而 D^2FA 的压缩比是 83.87%.显然,WD^2FA 的压缩能力要强于 D^2FA.此外,其中有 57 条规则,WD^2FA 与 D^2FA 具有相同的压缩能力,压缩比都是 93.82%.这说明了 WD^2FA 在最坏情况下与 D^2FA 具有相同的压缩能力,即 D^2FA 是 WD^2FA 在权值为 0 情况下的特例.

Table 7 Comparisons among number of transitions with WD^2FA, D^2FA, and DFA with ly-7 data

表 7 使用 ly-7 数据对 WD^2FA,D^2FA 和 DFA 状态转移数目的比较

Data information		Category of SM	# of transitions	Compression rate to DFA (%)
Total 113 Regex, average length is 56	57 Regex, WD^2FA and D^2FA have same compression capability	DFA	348 382	0
		D^2FA	21 534	93.82
		WD^2FA	21 534	93.82
	56 Regex, the compression capability of WD^2FA is better than D^2FA	DFA	464 643	0
		D^2FA	109 613	76.41
		WD^2FA	25 149	94.59
	Total 113 Regex	DFA	813 025	0
		D^2FA	131 147	83.87
		WD^2FA	46 683	94.26

为了进一步说明 WD^2FA 的压缩能力好于 D^2FA,从 Snort 规则集中取出 100 条正则表达式,并控制它们重复操作中长度限制的平均值(正则表达式{}内的数值),以控制正则表达式的复杂程度.平均长度限制越大,正则表达式的复杂度越高.长度限制分别取 1~12,状态转移数目的结果见表 8.

图 10 展示了 D^2FA 和 WD^2FA 相对于 DFA 状态转移数目的压缩比随着长度限制逐渐增长的变化曲线.图中,D^2FA 对应的曲线呈下降趋势,说明随着正则表达式复杂度的逐渐提高,D^2FA 的压缩性能在逐渐下降;而 WD^2FA 对应的曲线呈上升趋势,表明随着正则表达式复杂度的逐渐提高,WD^2FA 的压缩性能在逐渐上升.

Table 8 Number of transitions corresponding to different length restrictions

表 8 不同长度限制对应的状态转移数目

Category of FSM	Average of length restriction					
	1	2	3	4	5	6
DFA	400 900	445 200	480 000	517 135	577 280	637 440
D^2FA	64 400	83 700	105 590	130 005	155 855	188 050
WD^2FA	28 960	29 215	29 545	29 860	30 325	30 820
Category of FSM	Average of length restriction					
	7	8	9	10	11	12
DFA	704 000	776 960	860 160	957 440	1 073 910	1 214 720
D^2FA	222 825	261 460	305 230	355 425	413 345	479 030
WD^2FA	31 370	31 980	32 660	33 455	34 105	35 300

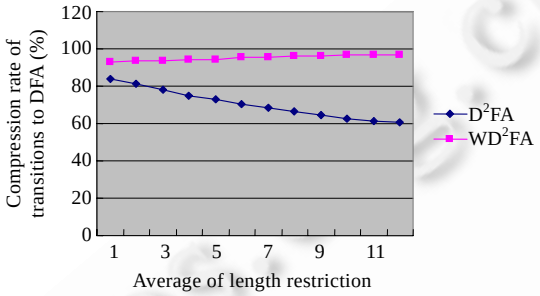


Fig.10 Effect on compression ratio performance of length restriction

图 10 长度限制对压缩比性能的影响

关于处理性能,WD^2FA 在扫描字符时相对于 D^2FA 的额外开销是经由默认转移时要对权值进行累加.除此之外,WD^2FA 与 D^2FA 一样,处理复杂度都取决于长度限制 L 的值,即处理一个字符在最坏情况下经由 L 次默认转移.因此,适用于 D^2FA 的性能改进算法也同样适用于 WD^2FA.

6.3 整体算法的压缩结果

依照存储算法整体结构的设计,对状态点数目压缩时,将 Snort 规则中的正则表达式根据转化目标进行分类,最终构建成 DFA 和 NFA 两类.然后,在此基础上对状态转移数目进行压缩,DFA 通过构建成 WD^2FA 进行压缩,NFA 用表压缩技术进行压缩,表 9 给出了压缩结果.通过 WD^2FA 算法和表压缩算法对复合的 FSM 的结合运用,得到状态转移数目的最终压缩比,约为 93%.

Table 9 Compression result of whole algorithms

表 9 整体算法的压缩结果

Category of Regex	Compression of states			
	Algorithms	# before comprssion	# after compression	Compression rate (%)
Constructiong for DFA	Table 6	To infinite	157 545	>99
Constructiong for NFA	NFA	To infinite	352 605	>99
Total size	—	To infinite	510 150	>99
Category of Regex	Compression of transitions			
	Algorithms	# before comprssion	# after compression	Compression rate (%)
Constructiong for DFA	WD^2FA	40 331 776	2 270 678	94.37
Constructiong for NFA	Table compression	88 942 417	7 086 111	92.03
Total size	—	129 274 193	9 356 789	92.76

7 结束语

本文从两个方面对 DFA 存储空间的压缩进行了探讨.

首先是针对状态点数目进行压缩.基于对正则表达式构建成 DFA 后,状态点数目的复杂度分析,提出了一种复合的 FSM 的构建方法,即结合了多种 DFA 压缩技术将不同复杂度的正则表达式分而治之,以使得所有平方级

和指数级复杂度的正则表达式对应的状态点数目控制到了线性级,从而使得用 DFA 实现基于正则表达式的模式匹配系统成为了可能.将来的工作可以对正则表达式复杂度分析进一步细化,或者根据将来的发展情况找出新的正则特征,同时在复合的 FSM 中也可以针对新的正则特征引入相应的 DFA 压缩技术,从而使得存储压缩和处理性能得到提高.

其次是针对状态转移数目进行压缩.本文引入了一种新的正则表达式的表示方法,即带权延迟 DFA,或者称为 WD^2FA,它在很大程度上减少了 DFA 的状态转移数目.通过实验可以发现,用 WD^2FA 压缩后的状态转移数目约为原来的 5%.对于不同复杂度的正则表达式,WD^2FA 压缩效果非常稳定,而且复杂度越高的正则表达式 WD^2FA 压缩效果越好.WD^2FA 的压缩性能好于 D^2FA,而且在最坏情况下具有与 D^2FA 相同的压缩性能,即 D^2FA 是 WD^2FA 在权值为 0 情况下的特例.在随后的工作中,可以设计和实现更高效的将 DFA 转化为 WD^2FA 的算法,从而降低构建 WD^2FA 的复杂度.

References:

[1] Aho AV, Corasick MJ. Efficient string matching: An aid to bibliographic search. Communications of the ACM, 1975,18(6): 333−340. [doi: 10.1145/360825.360855]

[2] Li WN, E YP, Ge JG, Qian HL. Multi-Pattern matching algorithms and hardware based implementation. Journal of Software, 2006, 17(12):2403−2415 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/2403.htm [doi: 10.1360/jos172403]

[3] Hopcroft JE, Motwani R, Ullman JD. Introduction to Automata Theory, Languages, and Computation. 3rd ed., Reading: Addison Wesley, 2006.

[4] Hopcroft J. An $O(n \log n)$ algorithm for minimizing states in a finite automaton. Technical Report, STAN-CS-TR-71-190, Stanford: Stanford University, 1971.

[5] Yu F, Chen ZF, Diao YL, Lakshman TV, Katz RH. Fast and memory-efficient regular expression matching for deep packet inspection. In: Bhuyan LN, Dubois M, Eatherton W, eds. Proc. of the 2006 ACM/IEEE Symp. on Architecture for Networking and Communications Systems. New York: ACM, 2006. 93−102. [doi: 10.1145/1185347.1185360]

[6] AbuHmed T, Mohaisen A, Nyang D. A survey on deep packet inspection for intrusion detection systems. Magazine of Korea Telecommunication Society, 2007,24(11):25−36.

[7] BrodieBC, Cytron RK,Taylor DE. A scalable architecture for high-throughput regular-expression pattern matching. In: Kaeli D, ed. Proc. of the 33rd Int'l Symp. on Computer Architecture. New York: ACM, 2006. 191−202. [doi: 10.1109/ISCA.2006.7]

[8] Becchi M, Crowley P. An improved algorithm to accelerate regular expression evaluation. In: Yavatkar R, Grunwald D, Ramakrishnan KK, eds. Proc. of the 2007 ACM/IEEE Symp. on Architecture for Networking and Communications Systems. New York: Association for Computing Machinery, 2007. 145−154. [doi: 10.1145/1323548.1323573]

[9] Kumar S, Dharmapurikar S, Yu F, Crowley P, Turner J. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In: Rizzo L, Anderson T, McKeown N, eds. Proc. of the 2006 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications. New York: Association for Computing Machinery, 2006. 339−350. [doi: 10.1145/1159913.1159952]

[10] Tuck N, Sherwood T, Calder B, Varghese G. Deterministic memory-efficient string matching algorithms for intrusion detection. In: Li VOK, Krunz M, Li B, eds. Proc. of the 23rd Annual Joint Conf. of the IEEE Computer and Communications Societies (IEEE INFOCOM), Vol. 4. Washington: IEEE Computer Society, 2004. 2628−2639. [doi: 10.1109/INFCOM.2004.1354682]

[11] Floyd RW, Ullman JD. The compilation of regular expressions into integrated circuits. Journal of ACM, 1982,29(3):603−622. [doi: 10.1145/322326.322327]

[12] Lin CH, Huang CT, Jiang CP, Chang SC. Optimization of pattern matching circuits for regular expression on FPGA. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 2007,15(12):1303−1310. [doi: 10.1109/TVLSI.2007.909801]

[13] Aldwairi M. Hardware-Efficient pattern matching algorithm and architectures for fast intrusion detection [Ph.D. Thesis]. North Carolina State University, 2006.

[14] Heering J, Klint P, Rekers J. Incremental generation of lexical scanners. ACM Trans. on Programming Languages and Systems (TOPLAS), 1992,14(4):490−520. [doi: 10.1145/133233.133240]

[15] Sommer R, Paxson V. Enhancing byte-level network intrusion detection signatures with context. In: Jajodia S, Atluri V, Jaeger T, eds. Proc. of the 10th ACM Conf. on Computer and Communications Security. New York: Association for Computing Machinery, 2003. 262−271. [doi: 10.1109/FPGA.2003.1227239]

[16] Green TJ, Gupta A, Miklau G, Onizuka M, Suciu D. Processing XML streams with deterministic automata and stream indexes. ACM Trans. on Database Systems, 2004,29(4):752−788. [doi: 10.1145/1042046.1042051]

[17] Chen D, Wong RK. Optimizing the lazy DFA approach for XML stream processing. In: Schewe KD, Williams H, eds. Proc. of the 15th Australasian Database Conf. Darlinghurst: Australian Computer Society, Inc., 2004. 131−140.

附中文参考文献:

[2] 李伟男,鄂跃鹏,葛敬国,钱华林.多模式匹配算法及硬件实现.软件学报,2006,17(12):2403−2415. http://www.jos.org.cn/1000-9825/17/2403.htm [doi: 10.1360/jos172403]

于强(1983－),男,河北唐山人,博士生,CCF 学生会员,主要研究领域为网络算法,生物信息学算法.

霍红卫(1963－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为并行算法,网络算法,生物信息学算法.