doi:10.3772/j.issn.1002-0470.2014.06.001

一种高效的正则表达式匹配方法①

张树壮② 吴志刚 罗 浩

(北京邮电大学网络技术研究院 北京 100876)

摘 要 为实现网络安全检测中大规模正则表达式的匹配,分析了在从非确定型有限自动机(NFA)到确定型有限自动机(DFA)的子集构造过程中导致状态爆炸性增长的原因,并提出了一种高效的正则表达式匹配方法。这种方法通过将部分 DFA 状态转变成受限的 NFA 状态来消除状态数量的剧烈增长,并会形成一种 DFA 状态与受限的 NFA 状态交替出现的有限自动机,称为 DNFA。DNFA 将 DFA 与 NFA 结合在一起,实现匹配速度与内存空间占用的平衡,其多层结构也更加适合复杂正则表达式规则。实验结果表明,上述方法可以在大大减少内存需求的情况下,实现正则表达式的高效匹配。

关键词 深度包检测,正则表达式,子集分割,有限自动机,混合自动机

0 引 言

当今,深度包检测技术已被广泛应用于网络安全的各个方面,例如入侵检测系统(intrusion detection system,IDS)、入侵阻断系统(intrusion prevention system,IPS)、应用层防火墙等。深度包检测技术是将一组给定的"模式"(又称为规则集)与每一个数据包的内容部分进行匹配。每一个"模式"表示一种攻击、恶意代码或者应用层协议的特征。匹配是计算机科学的一个经典问题,数十年来一直是研究的热点领域,也出现了很多优秀的研究成果,但是在网络安全领域,设计更加有效的数据结构和匹配方法,依然是个巨大的挑战,改进匹配效率,对提高整个系统的效率具有非常重要的意义。

在网络安全发展的初期,定义攻击和恶意代码等特征的规则通常是精确字符串,匹配算法也是经典的字符串匹配算法如 AC[1], WU-MANBER[2], SBOM[3]等。随着网络安全的发展和攻防双方对抗强度的增大,网络中恶意代码的数量也越来越多,使用精确字符串已经很难表示复杂而又庞大的特征。为了解决这一问题,从学术研究到商业化产品,都开始引入表达能力更强的正则表达式作为特征描述语言。正则表达式可以仅使用 1 条规则来描述一类"宽泛"的特征,从而免去对每一种具体特征的描述,因此,越来越多的正则表达式被应用到网络安全的各个方面,例如 Linux 应用层协议分类器(Linux Application Protocol Classifier,L7[4]),开源的入侵检测系统 Snort[5]、Bro[6],以及 Cisco 和 3Com 公司的商业化产品[7,8]等。除此之外,邮件过滤系统,面向应用层协议的过滤系统等,也都需要使用正则表达式对内容进行归类,从而实现特定的目标策略。随着正则表达式在网络安全领域中的广泛使用,近几年匹配技术的研究热点也从经典的字符串匹配转移到了正则表达式匹配。目前进行正则表达式匹配的典型方法有确定型有限自动机(deterministic finite automation,DFA)和非确定型有限自动机(nondeterministic finite automation,NFA)方法,但这两种方法都存在着匹配效率和内存需求之间不可调和的矛盾,无法胜任网络安全检测中大规模正则表达式的匹配。为了解决这一问题,本研究提出用一种新的混合自动机——DNFA 来进行正则表达式匹配。这种方法从自动机构建的角度对规则进行更加细粒度的分割,将 NFA 的思想融入到 DFA 的构建过程中,实现更加灵活的"时间-空间"复杂度平衡,并且具有更好的通用性。因此可以在使用适量内存的基础上,实现大规模正则表达式的高效匹配。

本研究给出了 DNFA 的构建方法并对其特点进

① 科技支撑计划(2012BAH37B02, 2012BAH42B02),863 计划(2012AA03001)和 242 计划(2013A012,2013A133)资助项目。

② 男,1982 年生,博士;研究方向:网络安全,信息内容安全;联系人,E-mail: zhangshuzhuang@bupt.edu.cn

(收稿日期:2012-12-18)

行了分析,对所提出的方法和其它代表性方法进行了实验对比,通过实验验证了该方法在实际中具有更高的应用价值。

1 相关工作

当前的绝大部分研究都是使用有限状态自动机(下文简称自动机)作为正则表达式匹配的工具。即首先将正则表达式编译成与其等价的自动机,然后将待检测数据流中的每一个字符作为输入来驱动自动机的遍历过程。基于自动机的匹配技术对"资源"的需求有两个方面:(1)保存数据结构所需要的存储资源;(2)执行匹配所需要的计算资源。

确定型有限自动机(DFA)对每个输入字符具有 $O(1)$ 的处理速度,适合在线的实时匹配,因此目前对正则表达式匹配技术的研究主要集中在 DFA 的匹配方法上。然而当多条正则表达式编译在同一个 DFA 中时,会引起内存需求的"爆炸性"增长[9],所以对正则表达式匹配的研究又都集中于如何降低 DFA 的存储空间上,目前的解决方法可以分为两大类:

第一类方法:通过消除"冗余"来消减 DFA 所占用的内存,代表性方法包括对输入进行预编码、合并转换函数、合并状态等。

文献[10]提出一种基于集合交割的方法来对输入字符进行预编码,文献[11]则提出了等价类分割的方法。这两种方法都是通过将整个字母表分成多个不相交子集,并使用子集来代替字母表中的元素,从而减少表的列数。文献[12]则进一步将状态和输入字符进行联合编码来提高对字母表的压缩效果。

2006 年 Kumar 等人在文献[13]中首先提出了 D^2FA 的方法,其基本思想是通过引入称为缺省转换(default transition)的转换函数来消除大量的等价转换表项。Kumar 还在文献[14]中对这种方法进行了改进,提出了 CD^2FA,以便在引入缺省转换的情况下也能通过访问一个状态完成对一个输入字符的处理。Becchi 在文献[11]中提出了另一种简单且有效的改进办法:使用状态的深度属性生成 D^2FA。在处理一个长度为 n 的字符串时,这种方法访问的状态数目不大于 $2n$。

2008 年 Ficara 等人在文献[12]中提出了 δFA 结构,这种结构中每个状态仅仅存储与其相邻状态不同的表项,而其余表项直接从其相邻点继承而来。文献[15]提出了状态合并(state merging)的方法。其基本思想是:如果两个状态具有目标相同的转换表项(无论是否对应于相同的字符),则将两个状态合并起来,形成一个混合状态,这样就大大减少了转换表的行数。文献[16]提出了转换表共享方法,用来改进状态合并方法。

上述冗余消除的方法虽然可以消减掉 DFA 中 90% 以上的转换边,但却无法彻底解决 DFA 的空间占用问题。因为 DFA 的状态数目可能呈指数级增长,而目前 DFA 内存缩减方法效果都为线性。并且这类方法需要首先完成 DFA 的构建才能进行缩减,而许多规则集根本无法在当前内存条件下完成构建。

第二类方法:在匹配中不再使用原始的 DFA 结构,而是使用 NFA 和 DFA 的混合结构或者改进的 DFA 结构,从而避免 DFA 状态的膨胀。代表性方法包括将正则表达式规则集进行分组(简称"规则分组")和构建混合自动机。

Fang 等人在文献[9]中提出了将正则表达式规则集进行分组的思想,通过将整个规则集编译成多个并行执行的 DFA 来减少由于表达式之间的互相影响而造成的状态增长(k-DFAs)。而文献[17]则提出一种基于模式的自动机 P-DFA 来进行分组,P-DFA 通过支持模式的动态增加和删除来快速评测将一条规则加入到某个规则组中的影响。对规则进行分组匹配实际上是在广义上使用了 NFA 的组织形式,只是每次跳转得到的活动状态集合中状态数目是一定的(等于分组的数目)。

后续的研究将这种方法进一步推广和细化,提出了混合自动机的思想。History-FA[18] 和 XFA[19,20] 引入了额外的信息来对 DFA 的匹配过程进行记录,从而大大减少了 DFA 为了模拟各种不同情况所需要的状态数目。但这两种方法都不能保证与原来 NFA 等价,而且在能处理的规则类型上有较大限制。文献[21]中提出了 Hybrid-FA,其特点就是首先将规则集编译成一个 NFA,然后只将一部分 NFA 转化成 DFA,形成 DFA 状态和 NFA 状态同时存在的情形,匹配时对两种不同的状态使用不同的匹配过程。Hybrid-FA 方法是通过对规则集进行分割来决定"确定性"与"不确定性"的边界,粒度比较大。

2 DNFA 的基本思想

以往的研究表明,如果正则表达式中出现了

“[X]*”([X]表示一组字符的集合,通配符“.”是[X]的一种特殊情况)连接符时,那么多条正则表达式编译在一起时会由于互相交互而引起 DFA 状态的增长,而当正则表达式中出现“[X]{m, n}({n}是{m, n}的一种特殊情况)”操作符时,单条表达式在编译时就可能会引起 DFA 状态的指数级增长。在实际使用的规则集中,这些操作符是大量存在的,因此在匹配时只能使用“规则分组”或者“混合自动机”等方法。

DFA 是由 NFA 通过子集构造法得到的,每个 DFA 状态等价于一组 NFA 状态的集合。如果用 D_{st} 表示 DFA 状态,N_{st} 表示 NFA 状态,则有

$$D_{st}=\{N_{st1},N_{st2},\cdots,N_{stk}\} \quad (1)$$

而状态增长现象的本质就是构建过程中 D_{st} 的跳转函数 δ 在字符 c 上的目标状态集合($\delta_{Dst}(c)$)总是一个新子集,即子集构造过程无法快速收敛。而无法收敛的原因就是 D_{st} 中包含 NFA 中用来处理“[X]*”的状态和模拟“[X]{m, n}”中计数的状态并且这些状态互相交互影响。

在实际使用的规则中,由于这些操作符频繁出现,因此在相应的 DFA 构建中,更多的是多种情况混合出现,从而使得构造过程更加复杂。为此,本文提出通过在子集构造过程中,对那些含有引起状态爆炸的 NFA 状态的子集进行分割来避免特殊状态之间的交互影响。但是对子集进行分割,必然会使得部分 DFA 状态对某个字符 c 的跳转函数 δ 具有多个目标子集,失去其“确定性”的性质从而退化成 NFA 状态。为了控制由于引入 NFA 状态而造成的匹配过程中性能的下降,本文规定将一个目标状态集合最多分割成 K 个子集,即自动机中每个 NFA 状态对于同一个字符 c,最多有 K 个转换函数,因此这类状态称为受限的 NFA 状态,用 c-N_{st} 表示。对于每个 c-N_{st} 而言,其跳转函数可以描述为

$$\delta_{c-Nst}(c):c-N_{st}\rightarrow S_1,S_2,\cdots,S_m,m\leqslant K \quad (2)$$

其中 $S_1\cup S_2\cup\cdots\cup S_m=\delta_{Dst}(c)$。

如果在子集构造过程中将部分 DFA 状态转化为受限的 NFA 状态,则会形成一个混合自动机。它由 DFA 状态和受限 NFA 状态混合组成,受限的 NFA 状态将整个自动机分成多层,如图 1 所示,因此称作 DNFA。DNFA 的第一层为 DFA 状态,而最后一层则可能是 DFA 状态,也可能是受限的 NFA 状态。

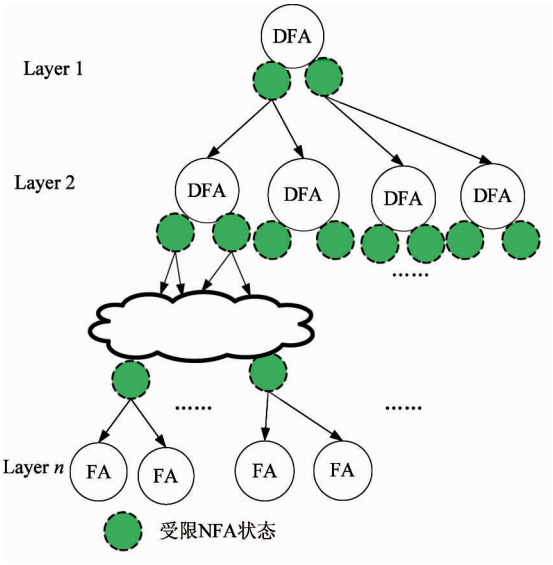


图 1　DNFA 的结构示意图

3　DNFA 的构建

上一节中分析了在从 NFA 转化为 DFA 的子集构造过程中引起状态增长的典型过程,并提出了受限 NFA 状态以及 DNFA 的概念,本节将介绍 DNFA 的构造方法并对其特点进行分析。

DNFA 由 NFA 转化而成,本文假设已经完成了从正则表达式规则到 NFA 的构建,则 DNFA 的构造过程如下:

(1)将 NFA 的初始状态作为 DNFA 的初始 DFA 状态加入到队列 q 中,设置当前 DNFA 的总层数 n 为 1。

(2)从队列 q 中取出一个状态 s,如果 s 是一个 DFA 状态,则对其执行标准的子集构造过程[22];如果 s 是一个受限的 NFA 状态,则将其对每个字符 c 的目标状态集合分割成 m($m\leqslant K$)个子集,形成 m 个跳转目标状态。如果整个 DNFA 的层数未超过指定值 N,则每个目标状态子集形成一个 DFA 状态,否则每个子集形成一个受限 NFA 状态。将生成的全部新状态加入到 q 中。

(3) 当第 n 层新构建的 DFA 数目达到一个指定值 M 时,将队列 q 中所有待处理的 DFA 状态转化为受限的 NFA 状态,并将 DNFA 的层数 n 增加 1。

(4)重复执行步骤(2)和(3),直到队列 q 为空。

生成的 DNFA 由 DFA 状态和受限的 NFA 状态组成,因此在匹配时只要对两种不同的状态分别进行处理即可。然而,如果在 DNFA 中一个受限的 NFA 状态的目标状态仍然是一个受限的 NFA 状态

时,就可能会使其在匹配过程中处理每个字符所需要访问的状态数目迅速增加,从而降低它的匹配速度。我们可以通过对构建过程进行优化来控制 DNFA 处理性能的下降。在构建过程中能够使得 DNFA 中受限 NFA 状态跳转向受限 NFA 状态的情况有两种:

第 1 种:在第(2)步中对一个受限 NFA 状态的跳转目标状态集合进行分割时,如果分割得到的子集与某个已存在的受限 NFA 状态等价,则会导致这种情况的发生。

第 2 种:当在第(3)步中将 DFA 状态转化为受限 NFA 状态时,如果当前的状态的父状态是一个受限 NFA 状态,也会出现这种现象。

为此,我们在构建过程中,加入如下两个优化策略:

(1)在对一个受限 NFA 状态 c-N_{st}的跳转目标进行分割时,如果得到的任何一个子集与已存在的受限 NFA 状态等价,则将此受限 NFA 状态 c $-N_{st}$重新转化为 DFA 状态,并对其执行标准的子集构造过程。

(2)记录每个 DFA 状态的父状态属性,如果其父状态是一个受限的 NFA 状态,则在第(3)步中,不将其转化为受限的 NFA 状态。

4 结果评测

高性能正则表达式匹配算法的主要评价指标有两个:内存需求和处理速度。本节将在实际规则集上给出本文方法在这两方面的性能评测结果,并分别与 Hybrid-FA 的结果进行比较和分析。

这两种方法都是以自动机为基础,因此我们用自动状态数目来评价其内存占用情况。但由于本文方法和 Hybrid-FA 使用的自动机都不是标准的 DFA 或 NFA,因此本文在对处理速度进行比较时,不但比对了其处理每个字符所需要访问的状态数目,还给出了相同硬件条件下的实际处理速度。

4.1 规则集特性

实验采用的正则表达式来自 Snort 入侵检测系统和 Linux Layer-7 filter(L7)。虽然 Snort 系统中有数千条正则表达式规则,但是它们并不会同时被启用,本文中选取了其中的两类规则:Web-misc 和 Backdoor。在表 1 中给出三个规则集各自的特性。

表 1 中第一列给出了每个规则集的规则数目,规模从 55 条递增到 150 条左右。第二列给出了规则集中 4 种连接操作符的出现次数。4 个用/号分开的数目顺次对应“. *”,“[X] *”, “.{m, n}”, “[X]{m, n}”。可以看出,在 Snort 规则集中,出现最多的是“[X] *”类型的操作符。特别是在 Backdoor 规则集中,这种操作符在每条规则中都出现将近 3 次。而 L7 规则集中前三种操作符出现的次数都比较多。第三列中给出了每个规则集中至少带有一个连接操作符的规则数目。可见,连接操作符在实际的规则集中是十分普遍的,特别是在 Snort 的两个规则集中,占了 90% 以上。最后一列则给出了用这些规则直接构建有限自动机时,所得到的 NFA 和 DFA 状态数目。表中 DFA 的状态数目为一个范围,因为三组规则在构建过程中都因为状态超过 10000000 而放弃构建,可见当表达式中包含有特殊连接符时 DFA 所需要的内存是十分巨大的。需要注意的是,虽然 Web-misc 规则集比较小,但是它所形成的 NFA 自动机数目是最多的,这是因为在 Web-misc 的规则中,出现了很多第 4 种类型(“[X]{n}”)的连接符,并且 n 的值都比较大(最大达到 1024),因而需要大量的 NFA 状态来模拟计数。

表 1 不同规则集的特性

规则集	规则数目	各种操作符的总数目	含有 4 种操作符之一的规则数目	NFA/DFA 状态数目
Web-misc	55	3/75/0/31	53	7615/ >10000000
Backdoor	153	9/388/0/10	130	3364/ >10000000
L7	89	23/33/23/3	57	1450/ >10000000

4.2 实验结果

本节中我们给出在不同规则集上构建各个方法的自动机所需要的内存,以及在不同测试数据集上的处理速度。为了进行全面的对比,本文采用了不同的参数来对每种方法的自动机进行构建。

对于 Hybrid-FA 方法,本文使用了不同大小的 head-DFA 进行了构建(head-DFA 的状态数目分别为:20000,40000,80000 和 160000)。在对 Web-misc

规则的构建过程中,采用了文献[21]中的方法对处于正则表达式末尾的"[X]{m, n}"型连接符使用了计数器进行优化处理。对于 DNFA,本文构建了状态数目约为 20000,40000 的 2 层自动机以及状态数目为 80000 的 2 层,3 层和 4 层自动机。在 Web-misc 规则的构建过程中也采用了文献[21]中的方法对处于正则表达式末尾的"[X]{m, n}"型连接符进行处理。本文实现所使用的数据结构每个状态大约需要 4KB 的内存。构建所需要的时间和最终得到的 DNFA 状态数目如表 2 所示。表中每一项的数值前面是状态数目,后面是构建所需要的秒数。

表 2　构建后的 DNFA 自动机状态数目

规则集	20000-2	40000-2	80000-2	80000-3	80000-4
Web-misc	20903/75	40136/160	82546/265	78310/251	80303/263
Backdoor	23642/88	42444/171	80640/257	79135/259	80427 /262
L7	31032/123	50295/212	85709/276	84005/272	82090/264

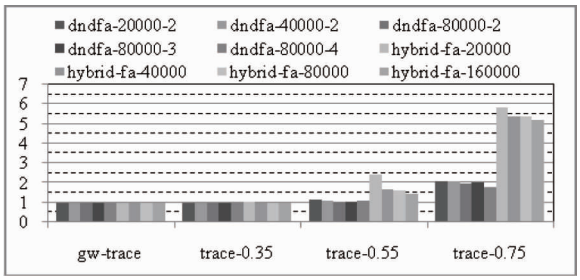
实验中所使用的测试数据包括从某企业网关上捕获的数据集(gw-trace)以及由 regex 工具[23]生成的测试 trace。这个工具可以为指定的正则表达式规则集生成具有一定特性的 trace 数据。为了做一个全面的对比,本文选取了 3 个不同的参数(p _ seed =0.35,0.55,0.75)为三个规则集分别生成了一组 trace 数据集。其中 p _ seed 表示在自动机的遍历过程中对相对当前状态更深层状态的访问概率。p _ seed 越大,访问深层状态的概率越大。

图 2、图 3 和图 4 分别给出了两种方法在三个规则集上的实验结果。其中图(a)给出了在处理一个字符平均需要访问的状态数目,图(b)给出了实际的处理速度。图中 Hybrid-FA-X 中的 X 表示总的状态数目;DNFA-X-Y 中的 X 表示总的状态数目,Y 表示自动机的层数。从图中可以看出:

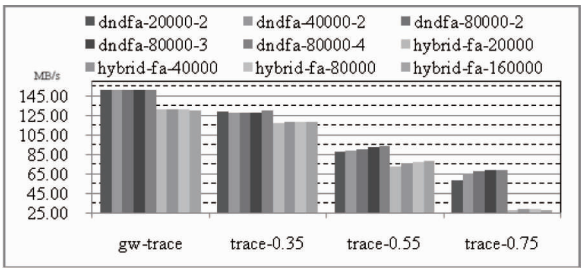
(1) Hybrid-FA 和 DNFA 方法都可以通过控制子集构造的进程来控制整个自动机的状态数目。但是由于 DNFA 在构造过程中进行了优化,因此在构建完成后的状态数目并不严格等于预设的状态数目。

(2) DNFA 比 Hyrbrid-FA 有更高的处理速度,并且 DNFA 和 Hybrid-FA 方法在 Web-misc 规则上的处理速度要低于相同条件下在 Backdoor 和 L7 规则上的处理速度。这是因为 DNFA 在构建过程中会跳过引起状态剧烈增长的部分继续确定化后面的部分,在状态数目相近时可以对更深的 NFA 状态进行确定化,因此在遍历时访问的 DFA 状态比例要大于 Hybrid-FA。在 Web-misc 规则集上,由于规则比较复杂,因此相同的总状态数目下两种方法能确定化的 NFA 层数要少于 Backdoor 和 L7,在匹配过程中遍历的 DFA 状态比例也就相对较小。而在通用处理器环境下对 DFA 状态的处理要比 NFA 状态的处理有更高的效率,因为在 DFA 状态下访问一个状态所需要的指令数要远远少于一个 NFA 状态所需要的指令数。此外,DNFA 在实际数据集 gw-trace 和 p _ seed 比较小的测试数据集上的处理性能要更好一些,因为被访问状态的平均深度越浅,DFA 状态所占的比例越大。

值得注意的是,网络安全中使用正则表达式作为匹配规则,是为了能够更准确地捕获各种攻击和恶意代码的特征,更加注重描述攻击行为的多个阶段或恶意代码多个部分之间的逻辑关系与位置关系。所以网络检测类的应用中所使用的规则一般都可以明显地分成若干个子特征,子特征之间用正则

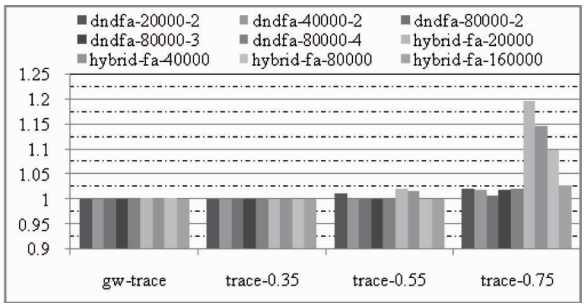


(a) web-misc 规则上处理一个字符平均访问的状态数目

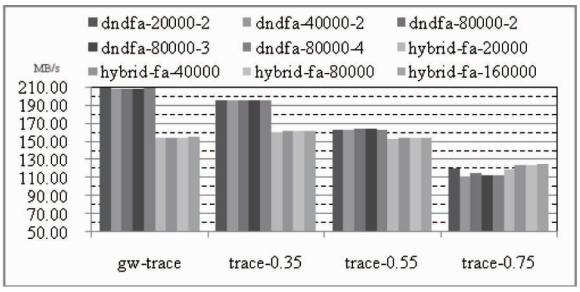


(b) web-misc 规则上实际处理速度

图 2　web-misc 规则上的处理结果

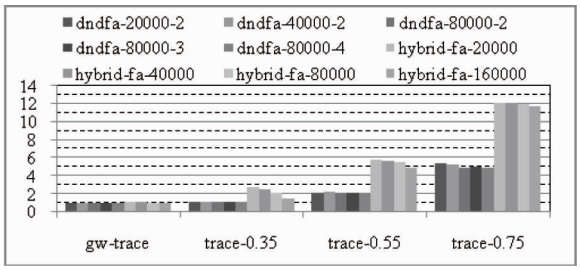


（a）backdoor 规则上处理一个字符平均访问的状态数目

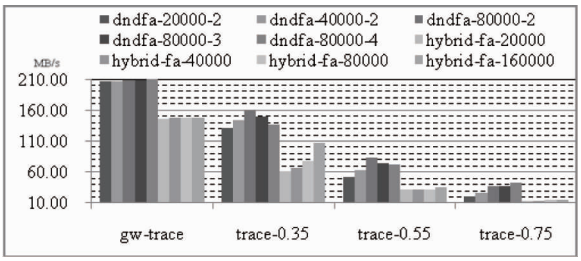


（b）backdoor 规则上处实际处理速度

图 3　backdoor 规则上的处理结果



（a）L7 规则上处理一个字符平均访问的状态数目



（b）L7 规则上处实际处理速度

图 4　L7 规则上的处理结果

表达式的运算符连接起来。因此 DNFA 的多层结构更加适合对网络应用中的规则进行匹配。

5　结 论

本文分析了将 NFA 转换为 DFA 的子集构造过程中特殊连接操作符引起状态数目增长的原因，并提出了对子集进行划分的方法和 DNFA 的定义。DNFA 是通过在子集构造过程中将部分 DFA 状态转化成受限的 NFA 状态而形成的一种分层的混合自动机。文中给出了 DNFA 的构建方法并对其性质进行了分析。DNFA 将 DFA 与 NFA 结合在一起，实现匹配速度与内存空间占用的平衡。实验结果表明，DNFA 的分层结构更加适合网络应用的各种规则集，具有更好的处理性能和适应性。

参考文献

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

[2] Wu S, Manber U. A fast algorithm for multi-pattern searching. Technical Report: TR-94-17, Department of Computer Science, University of Arizona, Tucson, AZ, 1994

[3] Allauzen C, Raffinot M. Factor Oracle of a Set of Words. Technical Report, Institute Gaspard-Monge, University, 1999:99-110

[4] Application Layer Packet Classifier for Linux. http://l7-filter.sourceforge.net/,2009

[5] Snort 2.8.x[EB/OL]. http://www.snort.org, 2009

[6] Bro Intrusion Detection System." http://bro-ids.org, 2010

[7] TippingPoint X505, http://www.tippingpoint.com/, 2009

[8] Cisco IOS IPS, http://www.cisco.com, 2010

[9] Fang Y, Zhifeng C, Yanlei D, et al. Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection. In:Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), San Jose, USA, 2006. 93-102

[10] 陈曙晖，苏金树，范慧萍等. 一种基于深度报文检测的 FSM 状态表压缩技术. 计算机研究与发展, 2008,42(8):1299-1306

[11] Becchi M, Cadambi S. An improved algorithm to accelerate regular expression evaluation. In: Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), Orlando, USA, 2007. 145-154

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

29-40

[13] Kumar S, Dharmapurikar S, Yu F, et al. Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection. In:Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM), Pisa, Italy, 2006. 339-350

[14] Kumar S, Turner J, Williams J. Advanced algorithms for fast and scalable deep packet inspection. In: Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), San Jose, USA, 2006. 81-92

[15] Becchi M, Cadambi S. Memory-Efficient Regular Expression Search Using State Merging. In: IEEE INFOCOM, Anchorage, USA, 2007. 1064-1072

[16] Zhang S Z, Luo H, Fang B X, et al. Fast and memory-efficient regular expression matching using transition sharing. IEICE transactions on Information and Systems, 2009, E92-D(10):1953-1960

[17] Jiang J C, Xu Y, Pan T, et al. Pattern-Based DFA for Memory-Efficient and Scalable Multiple Regular Expression Matching. In: Proceedings of the IEEE International Conference on Communications (ICC), Cape Town, South Africa, 2011. 1-5

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

[19] Smith R, Estan C, Jha S. XFA: Faster signature matching with extended automata. In:Proceedings of the IEEE Symposium on Security and Privacy, California, USA, 2008. 158-172

[20] Randy S, Cristian E, Somesh J. Deflating the big bang: fast and scalable deep packet inspection with extended finite automata. *Proceedings of the SIGCOMM Computer Communications Review*, 2008, 38(4):207-218

[21] Becchi M, Crowley P. A Hybrid Finite Automaton for Practical Deep Packet Inspection. In: Proceedings of the ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), New York, USA, 2007. 1-12

[22] Hopcroft J E, Motwani R, Ullman J D. An Introduction Automata Theory, Languages and Computation. 2nd ed., Boston: Addison Wesley, 2000. 1-50

[23] Becchi M, Franklin M, Crowley P. A Workload for Evaluating Deep Packet Inspection Architectures. In:Proceedings of the IEEE IInternational Symposium on Workload Characterization(IISWC), Seattle, USA, 2008. 79-89

An efficient regular expression matching method

Zhang Shuzhuang, Wu Zhigang, Luo Hao

(Institute of Network Technology, Beijing University of Post and Telecommunications, Beijing 100876)

Abstract

To realize the large-scale regular expression matching in network security inspection, the cause of the state "explosion" during the subset construction process from the nondeterministic finite automation (NFA) to the deterministic finite automation (DFA) is analyzed, and then the DNFA, an efficient regular expression matching method is proposed. This method avoids the dramatic growth of the states by transforming part DFA states into limited NFA states, thus the DNFA, a finite automation with the DFA state-limited NFA alternation, is formed. The DNFA takes advantage of the high processing efficiency of the DFA and the compact representation of the NFA to achieve a better trade-off between the memory space and the matching time. It can make a fine granularity splitting of rule set, and its multi-level structure is more suitable for complex regular expression rules in network applications. The experimental result shows that this proposal can provide a high throughout with a moderate memory requirement.

Key words: deep packet inspection, regular expression, subset splitting, finite automaton, hybrid finite automaton