doi:10.3969/j.issn.1000-436x.2014.06.007

基于两级存储的正则表达式匹配技术

陈曙晖，徐成成
（国防科学技术大学 计算机学院，湖南 长沙 410073）

摘 要：为解决正则表达式匹配中内存需求与检测性能的矛盾，首次提出两级存储的匹配方案。将马尔可夫链理论应用于自动机，通过求解稳态向量，得到各状态被随机访问的概率。将高概率的状态表项配置在 FPGA 嵌入存储器中，低概率的状态表项配置在 SRAM 中。使用 L7-filter 规则集进行实验，吞吐量达到 33 Gbit/s，匹配性能比将状态表完全存储在 SRAM 中提高了 50 倍。
关键词：正则表达式；马尔可夫链；两级存储；混合自动机
中图分类号：TP393.08 **文献标识码**：A **文章编号**：1000-436X(2014)06-0047-09

Regular expression matching technology with two-stage memory

CHEN Shu-hui, XU Cheng-cheng
(College of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract: To solve the contradiction between the memory requirement and the inspection performance, a matching engine with two-stage memory was proposed for the first time. To deploy the state table to two-stage memory, theories of Markov chain was applied to the FSA. By computing the steady vector, the random access probabilities of each state could be obtained. Further, the states with higher probabilities were deployed in the embedded memory of FPGA, and the states with lower probabilities were deployed in SRAM. Rules in L7-filter were tested in simulation experiments, and the results show that our method can reach a throughput of 33 Gbit/s in large scale FSA, which is 50 times than that of arranging the whole state table in SRAM.
Key words: regular expression; Markov chain; two-stage memory; hybrid FA

1 引言

随着网络的高速发展，网络的开放性导致的安全问题日趋严峻。网络安全的一个重要任务就是阻止入侵指令、病毒、木马等恶意信息流在网上传播，对报文进行深度检测是发现恶意信息流的有效手段。深度报文检测的原理是使用预定义的规则对流级报文负载进行匹配，这些规则是对恶意信息流的特征描述。深度报文检测技术广泛应用于入侵检测与防御、病毒检测、协议识别、网络取证等系统中。

早期的报文检测主要使用精确字符串匹配，一些经典的匹配算法如 AC、WU-MANBER、SBOM 可实现高速的报文匹配。但是精确字符串的表达能力非常有限，随着恶意代码复杂度的增加，特征的形式也变得越来越复杂，简单的字符串已远远不能满足描述的需求。正则表达式表达能力强，能够描述更为广泛的负载特征，已取代精确字符串成为深度报文检测的主要手段。研究领域和商业界都开始广泛采用正则表达式实现特征描述[1]。如 Linux 应用层协议分类器 L7-filter[2]、入侵检测系统 Snort[3]、Bro[4]基本上都使用正则表达式来描述规则，3Com 和 Cisco 的网络安全设备也都开始使用正则表达式。

在深度报文检测(DPI,deep packet inspection)中，首先要将正则表达式编译成有限状态自动机(FSA, finite state automata)，并将 FSA 的状态表配置在存储器中。匹配过程中，每处理报文的一个字节都需要至少一次查表，以获取下一次要访问的状态地址。匹配的速度取决于访存次数和每次访存的时延，而对于一个给定的报文，访存次数等于报文

收稿日期：2013-02-01；修回日期：2013-06-09

负载长度。因此要提高匹配的速度就需要尽量减少每次访存的时延。

在轻量级的网络下，网络链路速率低。如果规则数比较少，可以把每条规则编译成一个 FSA，状态表配置在高速存储器（如 FPGA 的片上存储器）中，以获得较高的匹配速度。然而，随着网络带宽的快速增加，10 Gbit 的网络已经开始应用于园区网络中；规则的数目也增加到数百甚至上千条。将每条规则编译成单个 FSA 的方案已经无法满足性能需求。如果将所有规则编译成一个 FSA，可能发生状态爆炸。状态表的规模可能超过数吉字节，远超过目前高速存储器的容量，只能配置在 DRAM 这样的低速存储器中，访存时延大大提高。

针对正则表达式匹配所带来的存储空间膨胀问题，目前的研究都集中于压缩状态表以减小内存需求。通常这样的压缩是非常有限的，即使压缩率能达到 90%以上，压缩后状态表的空间仍然远大于目前高速存储器容量。状态表只能配置在大容量的低速存储器中，匹配速度严重受限于低速存储器的访存时延。内存需求和匹配性能是一对相互制约的因素。

为解决匹配性能与存储空间的矛盾，本文从另一个角度出发，首次提出了一种两级存储的匹配技术，结合使用小容量的高速存储器和大容量的低速存储器，获得高速的匹配性能。基本思想为：首先基于状态表生成状态转移概率矩阵，用马尔可夫链理论求得状态转移概率矩阵的稳态向量，进而得到每个状态被随机访问的概率；其次设计了一个两级存储的匹配引擎，根据状态被随机访问概率的大小，将高概率的状态表项配置在高速存储器中，低概率的状态表项配置在低速存储器中；最后为验证上述方法的可行性，使用 L7-filter 规则集进行仿真实验，使用 DARPA99 的报文及实际捕获的网络报文分别进行匹配。对匹配过程中的访存情况进行统计分析，实验证明两级存储技术能极大地提高匹配性能。

2 相关工作

传统上有 2 种类型的 FSA：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)。DFA 在任意时刻只有一个活跃状态，对任意一个输入字符只产生一次状态转移，只需要访存一次。而 NFA 可能同时有多个活跃状态，理论上最大活跃状态数等于 NFA 状态数，处理一个输入字符可能需要多次访存，同时还需要处理 ε 转移。另外将 m 条正则表达式编译成单个 DFA 时，可以将处理复杂度从 $O(m)$ 降到 $O(1)$，DFA 的时间复杂度明显优于 NFA[5]。因此实践中通常采用 DFA 作为自动机，但随着规则数量的增加及复杂化，组合的 DFA 可能导致状态数爆炸，理论上最大状态数可达到 $O(\sum^{mn})$，其中，n 为每条规则长度，很多常用的规则集在一般的系统中甚至无法生成完整的 DFA。因此目前的研究主要集中在 2 个方面：1) 压缩状态表以减少 DFA 的内存需求；2) 设计新型的自动机以解决状态爆炸的问题。

在压缩状态表方面，Becchi 提出合并状态的压缩方法[6]，通过在输入输出的转换边上引入标记来合并不相等的状态。陈曙晖提出合并输入字符的压缩方法[7]，在正则表达式转换成 DFA 之前，对输入符号进行集合交割的预编码，通过压缩输入减少输入符号的种类，从而压缩状态表的空间。Kumar 提出减少转换边的压缩方法 D^2FA [8]，不断地将 DFA 的多条转换边用一条缺省转换边来代替，对状态表的压缩率可达到 95%。杨毅夫提出簇分割的方法[9]，将 DFA 分割成 3 个部分存入 3 个矩阵中，并分别对 3 个矩阵进行压缩。

在解决状态爆炸方面，YU F 提出一种朴素的分类方法[5]，将表达式划分成多个不相交的集合，每个集合单独编译成一个 DFA。Sommer 提出动态生成 DFA 的方法[10]，在匹配过程中动态地将匹配路径上的 NFA 转换成 DFA，其他 NFA 状态保持不变。Kumar 提出基于历史状态的自动机[11]，存储附加信息以辅助自动机匹配，以此减少生成的 DFA 状态数。Becchi 提出混合自动机(HFA, hybrid-FA)的概念[12]，根据设定的阈值将部分 NFA 状态转换成 DFA，在减少内存需求的同时获得近似于 DFA 的匹配性能。

另外，Lin 提出一种多线程自动机 M-DFA[13]，在多核系统上并行匹配的同时对状态表进行压缩。王磊等[14]利用 GPU 高并行的特点，实现了一种高速的正则表达式匹配引擎。

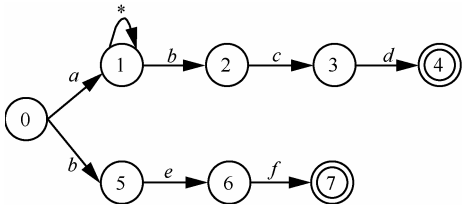
本文的研究从另一个角度出发来解决问题，但以上压缩状态表的算法都可以应用于本文的低速存储器中以减少内存需求。首先为编译得到完整的自动机，需采用新型结构的自动机。本文采用 Becchi 提出的 HFA，因为它结构简单，不需要存储额外的

辅助信息，同时兼具 DFA 和 NFA 的优点。

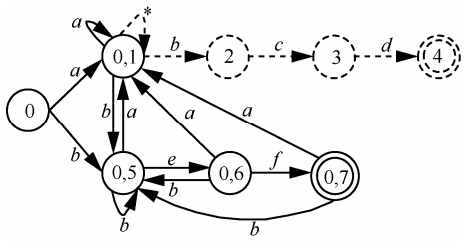
DFA 的状态爆炸主要由 2 种情况引起[5]，一种是带重复次数限定符如“{*m*,*n*}”或“{*n*}”的规则，编译时需要考虑各种可能的排列组合情况，有时即使编译单条这种类型的规则也可能产生状态爆炸。另一种是带通配符如“.*”的规则，在编译这种规则时将完整地复制其他规则编译成的自动机。编译单条此类型的规则不会产生状态爆炸，多条此类型的规则一起编译时会大大提高自动机的复杂度。

HFA 的基本思想是在生成 NFA 后并不将其完全转换成 DFA。而是在子集构造算法中根据当前 NFA 状态集合中各个 NFA 状态的类型来决定是否将其确定化，如果某个 NFA 状态是以上 2 种类型生成的，则不将这个 NFA 状态确定化。含有未确定化 NFA 状态的 DFA 状态称为边界状态，边界状态可以作为 DFA 的一部分或者 NFA 的一部分。生成的 HFA 包含一个头部的 DFA，若干边界状态，每个边界状态指向若干 NFA 状态，图 1 是 HFA 的一个简单示例，子集构造算法在 NFA 状态 1 处不再继续生成 DFA。

图 1(b)的 HFA 中，虚线圆表示的状态为 NFA 状态，虚线箭头表示 NFA 的状态转换，实线圆表示的状态为 DFA 状态，其中，DFA(0,1)为边界状态。当在边界状态时，NFA 状态 1 为 NFA 部分的活跃状态。此时输入字符 *b*，DFA 的活跃状态为 DFA(0,5)，NFA 活跃状态为 NFA 状态 1 和 NFA 状态 2。实际上在表达式比较多的时候，生成的 HFA 通常含有多个边界状态，每个边界状态可能指向多个 NFA。HFA 有以下特征：1）初始状态是 DFA 状态；2）在访问边界状态时，相应的 NFA 状态才会被激活；3）没有从 NFA 状态到 DFA 状态的转换。在匹配的过程中，DFA 部分始终只有一个活跃状态，而 NFA 活跃状态可能会增加、减少，甚至消亡。



(a) 2 条规则生成的 NFA



(b) 2 条规则生成的 HFA

图 1　规则“*.*a.*bcd*”和“*.*bef*”生成的 NFA 及相应的 HFA

HFA 能有效压缩状态表空间，但没有结合当前器件特性解决匹配性能问题。因为虽然 HFA 对存储器的需求大大减小，但是对整个 L7 filter 规则集生成的状态数仍有数十万条，从而导致整个存储空间达到近百兆的规模，无法将整个状态表存储在高速存储器中，匹配性能依然不高。

3　正则表达式匹配处理

正则表达式匹配面临的主要挑战是内存需求与匹配性能的矛盾，高速存储器容量不够大，无法存储整个状态表，而大容量存储器的匹配性能又不高。下面通过一个简单的示例对自动机的匹配速度进行分析，然后对主流的存储器进行对比，最后提出作者的设计思想。

以图 1 中的 HFA 为例，假设用输入字符串“*babef* ”进行匹配。匹配过程如表 1 所示。

表 1　　字符串“*babef*”的匹配过程

状态	*b*	*a*	*b*	*e*	*f*
DFA	(0,5)	(0,1)	(0,5)	(0,6)	(0,7)
NFA		1	1、2	1	1

匹配结果是 DFA 状态(0,7)和 NFA 状态 1，DFA 状态(0,7)是接受状态，因此匹配成功。从上述示例可以看出，每处理报文的一个字节都至少需要访问一次存储器。下一次要访问的地址取决于上一次访问的地址和将要处理的字符。连续 2 次的访存操作是相关联的，因此无法使用访存的流水线技术。实践中对 HFA 的匹配过程主要是访问 DFA 部分，即访存的次数基本上等于报文长度，因此在给定报文的情况下，匹配吞吐量取决于每次访存的时间。要提高匹配性能就必须减少每次访存的时延，最好的方法是使用更加高速的存储器来存储状态表。

然而随着规则数目的增加及复杂化，将多条正则表达式编译成一个自动机时内存的需求是很大的。尤其当表达式中含有大量的通配符“.*”及重复次数限定符“{*n*}”时，自动机的状态数可能使系统内存无法承受。尽管有很多压缩技术可对状态表进行压缩，但线性的压缩根本无法解决问题。虽然可

以采用新型的自动机如 HFA 对状态数进行控制，但是状态空间仍然远超过目前高速存储器的容量。

不同类型的存储器性能差别很大。例如 SRAM 有很高的访问速度，目前已经可以达到 250 MHz，但是它的价格昂贵且容量不大，一般只有几兆字节，只能存储几千条状态。DRAM 有高吞吐量高容量的特点，但是它有一个很长的启动时间，一次访存需要花费上百个周期才能获得结果。在正则表达式的匹配处理中，连续 2 次访存之间是相互关联的，下一次访存的地址取决于上一次访存的结果，因此流水线技术和 DRAM 的高吞吐量无法得到充分利用。FPGA 有高吞吐量的片上存储器，并且可通过双端口并行访问，其访存时间只有 1~2 ns，非常适合于连续依赖的访存。但是片上存储器的容量很小，一般不超过 10 Mbit，只能存储数百条状态。

虽然 FPGA 的片上存储器容量很小，但是它的最大频率已经超过 500 MHz，即使综合后频率可能只有 400 MHz，仍然远高于 DRAM 和 SRAM。以 Altera Stratix II EP2S180 为例，它有 9 Mbit 的 RAM 内存块，并且每个内存块可以配置成双端口的模式。假设综合后的频率是 400 MHz，仅含一个扫描模块的匹配引擎的吞吐量可以达到 400 MHz×2×8= 6.4 Gbit/s。实际上将这些内存块的访问并行化后可以达到超过 20 Gbit/s 的吞吐量，非常适合于高速正则表达式匹配。

通过以上分析可知，使用单一的存储器无法同时满足容量和性能的需求。在计算机体系结构中，Cache 是用于 CPU 和主存之间的高速小容量存储器。它存储那些经常被访问的程序或数据的副本供 CPU 访问，以减少访问主存的时延。只要访存尽可能多地在 Cache 中进行，CPU 整体访存性能就接近于 Cache 访存性能。借鉴此思想，考虑将状态表中高访问概率的状态存储于高速存储器中，低访问概率的状态存储于低速存储器中，通过适当的分配可以达到性能与空间的完美结合。

在本文的设计中，选择 FPGA 片上存储器作为一级存储器，因为片上存储器速度很快，可以达到 400 MHz。另外主流的 FPGA 都有若干个可以同时被访问的存储块。如果将相同的内容存储在不同的存储块中，访存操作可以在不同的存储块中并发进行，吞吐量可以提高数倍。由于 SRAM 控制电路简单，通过多个 SRAM 集中控制足以容纳整个状态表。

正则表达式分级存储的关键是解决高速存储器的命中率问题。在计算机系统中，由于局部性原理，可以用先进先出、最近最少使用等替换算法，使 Cache 有较高的命中率。但是在深度报文检测中，报文的内容却是完全随机的，无法预知下一个要处理的字节内容，会转向哪个状态。选择那些经常被访问的状态存储到高速存储器中是提高性能的关键。

在 HFA 中，因为 NFA 部分只有在到达边界状态时才会被激活。通常情况下，到达边界状态时概率已经非常小，所以 NFA 部分被访问的概率远小于 DFA 部分被访问的概率。因此可以把注意力集中在 HFA 的 DFA 部分，从中找出访问概率较高的状态。在 DFA 部分，自动机接受一个确定的字符，转向一个确定的状态。下一步要转移的状态只与当前的状态有关，而与以前任意时刻它所处的状态都无关，这是典型的马尔可夫过程。可以用随机过程中的马尔可夫链的方法来解决状态访问概率的问题。

4 状态概率

首先引入马尔可夫链的思想，马尔可夫链是随机变量 $X_1, X_2, X_3, \cdots$ 的一个数列。这些变量的范围即它们所有可能取值的集合，称为“状态空间”，而 X_n 的值则是在时间 n 时所处的状态。对任意的 n，如果 X_{n+1} 对过去状态的条件概率分布仅是 X_n 的函数，即 $P(X_{n+1}=x|X_1, X_2, \cdots, X_n)=P(X_{n+1}=x|X_n)$，则这样的序列具有马尔可夫性质[15]。

假定报文内容是随机的，那么报文匹配过程中的状态迁移就是一个随机过程。而下一步要转移的状态只与当前状态和输入字符有关，与之前任意时刻所处的状态都无关，具有典型的马尔可夫性。由于自动机是固定的，任意时刻状态间转移概率不变，因此又具有齐次性[15]。

在齐次马尔可夫链中，给定一个初始概率分布，假定每一步的转移都是按概率随机的，当转移次数趋于无穷时，各个状态出现的概率趋于一个定值，各状态出现概率组成的向量称为稳态向量[16]。状态分布概率达到稳态向量 $\boldsymbol{s}$ 时，意味着经过一步随机的状态转移，各状态分布概率的向量仍然是 $\boldsymbol{s}$。即各个状态的分布概率不再随时间发生变化。并且稳态向量与初始的状态分布概率无关，只与状态之间的转移关系有关。事实上在转移次数比较大的时候，每个状态出现的概率就已经趋于稳定。

在 DFA 部分的报文匹配是一个典型的齐次马尔可夫过程，而报文内容可以认为是完全随机的，因此可以将这个马尔可夫过程的稳态向量作为各

个状态被随机访问的概率，从而得到访问概率较高的 DFA 状态。获取各个状态被访问的概率的步骤如下：1）计算转移概率矩阵；2）求解稳态向量；3）状态重命名。其中状态重命名是为了降低一级存储器寻址的复杂度。

4.1　计算转移概率矩阵

要求解DFA状态的稳态向量，首先要根据DFA求转移概率矩阵 $\boldsymbol{P}$。通常 DFA 可以表示成一个五元组的形式 (Q,Σ,δ,q_0,A)，其中 Q 表示 DFA 的状态集，Σ 为输入字符表（包含 256 个字符），δ 为转移函数 $Q\times\Sigma\to Q$，q_0 为初始状态，A 为接受状态集。转移概率矩阵 $\boldsymbol{P}$ 是一个 $|Q|\times|Q|$ 的方阵，$\boldsymbol{P}$ 中任意一个元素 P_{ij} 代表状态 i 随机转移到状态 j 的概率。

在 DFA 中，假设匹配引擎当前所处的状态为 q_i，如何求解它一步转移到状态 q_j 的概率？如果不存在字符 a 满足 $\delta(q_i,a)=q_j$，那么从 q_i 转移到 q_j 的概率为 0；如果存在字符 a 满足 $\delta(q_i,a)=q_j$，由于 DFA 中 $\delta(q_i,a)$ 唯一，并且报文内容是随机的，即字符表中每个字符出现的概率是相等的，那么从 q_i 转移到 q_j 的概率与字符表中满足 $\delta(q_i,a)=q_j$ 的字符 a 的个数成正比。例如，当前状态 q_i 接收一个输入字符只能转移到状态 q_j 或 q_k，满足 $\delta(q_i,a)=q_j$ 的字符集为 B，满足 $\delta(q_i,a)=q_k$ 的字符集为 C，则从 q_i 转移到 q_j 和 q_k 的概率分别为 $|B|/|\Sigma|$ 和 $|C|/|\Sigma|$，转移到其他状态的概率均为 0。

进而可以通过状态表 $|Q|\times|\Sigma|$ 计算一步转移概率矩阵 $\boldsymbol{P}$，其中，第 i 行第 j 列的元素 p_{ij}：$\left|\left\{a|\delta(q_i,a)=q_j,a\in\Sigma,q_i,q_j\in Q\right\}\right|/|\Sigma|$。由于假定报文内容是随机的，$\boldsymbol{P}$ 的每个元素的值介于 0 和 1 之间，并且每行元素之和为 1。以规则“.*ab+ac”为例，对应的 DFA 如图 2 所示。

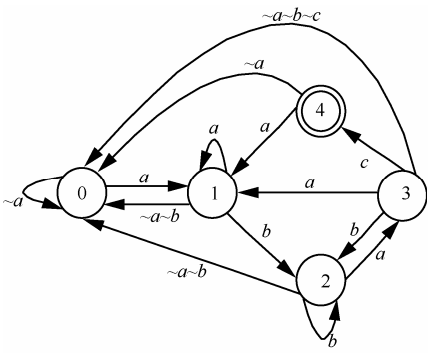


图 2　规则“.*ab+ac”的 DFA

其转移概率矩阵为

$$\boldsymbol{P}_1=\begin{pmatrix}\frac{255}{256} & \frac{1}{256} & 0 & 0 & 0\\ \frac{254}{256} & \frac{1}{256} & \frac{1}{256} & 0 & 0\\ \frac{254}{256} & 0 & \frac{1}{256} & \frac{1}{256} & 0\\ \frac{253}{256} & \frac{1}{256} & \frac{1}{256} & 0 & \frac{1}{256}\\ \frac{255}{256} & \frac{1}{256} & 0 & 0 & 0\end{pmatrix} \tag{1}$$

从式(1)中可以看出，每个状态转移到其他状态的概率，例如状态 0 接收字符 a 转移到状态 1，接收其他 255 个字符转移到自身，所以 $p_{00}=255/256$，$p_{01}=1/256$。现在已经通过 DFA 状态表得到一步转移概率矩阵 $\boldsymbol{P}$，下面求解稳态向量。

4.2　求解稳态向量

进一步求解稳态向量之前，需要对马尔可夫链的性质进行讨论。从马尔可夫链的任意一个状态出发，经过有限步转移，能够遍历所有的状态，这样的马尔可夫链是不可约马尔可夫链[17]，否则为可约马尔可夫链。直观地讲，不可约马尔可夫链中所有的状态都是相通的、可以互达的。例如，图 2 所示的 DFA 就是一个简单的不可约马尔可夫链，从任意一个状态出发都能到达 DFA 的 5 个状态。对于不可约马尔可夫链的转移概率矩阵 $\boldsymbol{P}$，存在一个自然数 m，当 $n\geqslant m$ 时，$\boldsymbol{P}^n$ 不再变化，即 $\lim\limits_{k\to\infty}\boldsymbol{P}^k=\boldsymbol{P}^m$，称 $\boldsymbol{P}^m$ 为稳态矩阵。并且 $\boldsymbol{P}^m$ 的每一行都相等，都等于稳态向量 $\boldsymbol{s}$。另外由于稳态向量不再随时间变化，显然有 $\boldsymbol{Ps}=\boldsymbol{s}$。例如式(1)中 $\boldsymbol{P}_1$ 的稳态向量是$(9.9\times10^{-1}, 3.9\times10^{-1}, 1.5\times10^{-5}, 5.9\times10^{-8}, 2.3\times10^{-10})$，这样就得到各状态被访问的概率。

然而在深度报文检测系统中，规则比较复杂，得到的马尔可夫链通常是可约的，即从某些状态出发无法到达所有的状态。尤其是当规则中带有首锚时，例如，规则“^a.*d”和规则“.*b?c”编译得到的 DFA 如图 3 所示。

很明显，这是一个可约马尔可夫链，从状态 0 之外任何一个状态都不能到达状态 0。

在可约马尔可夫链中，可以根据状态之间的可达性将整个状态空间划分成过渡状态集和闭集，闭集可能不止一个。同一个闭集内任意 2 个状态是相互可达的，从一个闭集中的状态出发，不能到达该

闭集外任意其他状态。例如图 3 中{0}是过渡状态集，{1,3}和{2,4,5}是闭集。当状态数比较多时，可以用 Xie 提出的有限状态马尔可夫链状态空间划分算法[18]对状态集进行划分。

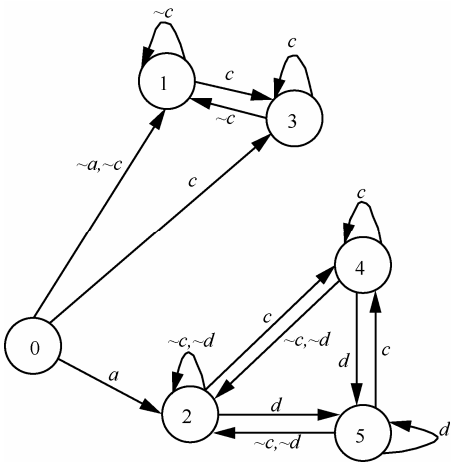


图 3　规则“^a.*d”和“.*b?c”的 DFA

下面主要研究可约马尔可夫链的稳态向量。可约马尔可夫链通常由许多闭集组成，匹配时从状态 0 开始，状态 0 是过渡状态，从 0 出发可以到达任意的闭集。从匹配的角度来看，不带首锚的规则通常在闭集中匹配成功，带首锚的规则可能在过渡状态集或闭集中匹配成功。为方便处理，对初始状态概率矩阵进行行列变换[16]得到形如(2)的转移概率矩阵。

$$\boldsymbol{P}=\begin{bmatrix} \boldsymbol{T} & \boldsymbol{A}_1 & \boldsymbol{A}_2 & \cdots & \boldsymbol{A}_l \\ 0 & \boldsymbol{C}_1 & 0 & 0 & 0 \\ 0 & 0 & \boldsymbol{C}_2 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & \boldsymbol{C}_l \end{bmatrix} \tag{2}$$

其中，$\boldsymbol{T}$ 表示过渡状态集的转移概率方阵，$\boldsymbol{A}_i$ 表示过渡状态集转移到第 i 个闭集的非负概率矩阵，$\boldsymbol{C}_i$ 表示第 i 个闭集的随机转移概率矩阵。可以看出，任意闭集转移到该闭集外状态的概率是 0，过渡状态集有向闭集的转换。通过矩阵求幂可以得到 $\boldsymbol{P}$ 的 n 步转移概率矩阵为

$$\boldsymbol{P}^{(n)}=\begin{bmatrix} T^{(n)} & Y_1^{(n)} & Y_2^{(n)} & \cdots & Y_l^{(n)} \\ 0 & C_1^{(n)} & 0 & \cdots & 0 \\ 0 & 0 & C_2^{(n)} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & C_l^{(n)} \end{bmatrix} \tag{3}$$

当 $n\to\infty$ 时，$\boldsymbol{T}^{(n)}\to 0$，并且有[16]

$$Y_i^{(n)}=\sum_{k=0}^{n-1}T^{(n-k-1)}\boldsymbol{A}_i\boldsymbol{C}_i^k=(I-T)^{-1}\boldsymbol{A}_i\boldsymbol{C}_i^n \tag{4}$$

对于可约马尔可夫链同样存在一个稳态矩阵 $\boldsymbol{W}$，$\boldsymbol{W}=\lim_{n\to\infty}\boldsymbol{P}^{(n)}$。但是不同于不可约马尔可夫链，$\boldsymbol{W}$ 并不是每一行都相等，只有同一个闭集中的状态所在的那些行是相等的。因为 $\boldsymbol{C}_i$ 可以看作是一个不可约马尔可夫链，它的稳态矩阵中各行是相等的。虽然 $\boldsymbol{W}$ 每行并不相等，但这并不影响对稳态向量的求解，因为自动机的匹配总是从状态 0 开始的，所以只需要得到状态 0 所在行的稳态向量即可。

有 2 类方法求解稳态矩阵 $\boldsymbol{W}$，第一类是用迭代法求解线性方程组 $\boldsymbol{WP}=\boldsymbol{W}$，如雅可比迭代、高斯—赛德尔迭代、超松弛迭代[19]等；第二类是求极限的方法直接求取稳态矩阵[20]。对于第一类方法，由于需要的只是状态 0 所在行的稳态向量，所以只需要求解方程组

$$\begin{aligned}[w_1w_2\cdots w_m](\boldsymbol{P}-\boldsymbol{E})&=0\\ w_1+w_2+\cdots+w_m&=1\end{aligned} \tag{5}$$

其中，$[w_1w_2\cdots w_m]$ 为某一行对应的稳态向量。用迭代法求解方程组的时间复杂度为 $O(m^3)$，在规则集比较复杂的时候，$\boldsymbol{P}$ 的阶数可能达到数百万，求解时间很长。另外即使能够求出一些稳态向量的解，还需要额外的计算来确定哪个解是状态 0 所在行对应的稳态向量。虽然在一些特殊情况下，例如 $\boldsymbol{P}$ 是三角矩阵或带状矩阵时，可以有一些特殊的方法来求解[16]，但通常情况下得到的转移概率矩阵是没有显著特征的。另外确定停止迭代的条件也是一个很复杂的问题，所以第一类方法不适合求解大规模转移概率矩阵的稳态向量。

如果用直接求极限的方法求解稳态向量，需要分别求解 $\boldsymbol{Y}_i^{\infty}$ 和 $\boldsymbol{C}_i^{\infty}$。从式(4)可知，$\boldsymbol{Y}_i^{\infty}$ 的求解依赖于 $\boldsymbol{C}_i^{\infty}$。如果每个闭集 $\boldsymbol{C}_i^{\infty}$ 的规模都比较小，$\boldsymbol{C}_i^{\infty}$ 可以很快地通过矩阵的幂运算得到。再利用式(4)计算 $\boldsymbol{Y}_i^{\infty}$，进而得到稳态矩阵 $\boldsymbol{W}$，$\boldsymbol{W}$ 第一行元素即为从状态 0 出发的稳态向量。实践证明，通常得到的闭集规模都比较小，只有数百阶，因此适合用第二类方法求解稳态向量。另外矩阵相乘可能会带来精度的影响，但这并不影响最终的结果，因为需要的是稳态时每个状态概率的相对大小而不是绝对大小。

以图 3 为例，首先用集合划分算法将状态集划分成过渡状态集{0}和闭态集 $\boldsymbol{C}_1${1,3}、$\boldsymbol{C}_2${2,4,5}，通过行列变换得到转移概率矩阵 $\boldsymbol{P}_2$ 为

$$\boldsymbol{P}_2=\begin{pmatrix} 0 & \frac{254}{256} & \frac{1}{256} & \frac{1}{256} & 0 & 0 \\ 0 & \frac{255}{256} & \frac{1}{256} & 0 & 0 & 0 \\ 0 & \frac{255}{256} & \frac{1}{256} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{254}{256} & \frac{1}{256} & \frac{1}{256} \\ 0 & 0 & 0 & \frac{254}{256} & \frac{1}{256} & \frac{1}{256} \\ 0 & 0 & 0 & \frac{254}{256} & \frac{1}{256} & \frac{1}{256} \end{pmatrix} \tag{6}$$

进而求得 $\boldsymbol{C}_1^{\infty}=\begin{pmatrix} 0.996 & 0.004 \\ 0.996 & 0.004 \end{pmatrix}$

$$\boldsymbol{C}_2^{\infty}=\begin{pmatrix} 0.992 & 0.004 & 0.004 \\ 0.992 & 0.004 & 0.004 \\ 0.992 & 0.004 & 0.004 \end{pmatrix}$$

$$\boldsymbol{Y}_1^{\infty}=(0.992\,20, 0.003\,89)$$

$$\boldsymbol{Y}_2^{\infty}=(0.003\,87, 0.000\,02, 0, 000\,02)$$

排序得到从状态 0 开始的稳态向量中闭集合状态概率排序为 1，3，2，4，5。

通过 4.2 节的方法可以得到稳态时各状态访问概率。实际上得到的只是闭集中状态的概率，因为过渡状态集中所有状态概率为 0。现在可以根据一级存储器空间大小，将 DFA 部分的访问概率较高的一部分状态配置在一级存储器中，HFA 的整个状态表存储在二级存储器中。基于两级存储的匹配引擎结构如图 4 所示，图中省略了 SRAM 控制器模块。其中，虚线框内的部分代表 FPGA 部分，Cache 为 FPGA 片上存储块，SRAM 为二级存储器，如果需要可以在此基础上增加 DRAM 为三级存储器。Cache 中状态表项是固定配置的，因此 Cache 不需要和 SRAM 连接。多块 Cache 的目的是并发访问，提高吞吐量；多块 SRAM 目的是解决状态表存储空间问题。匹配过程中，如果要查找 DFA 状态，则优先访问 Cache，Cache 未命中时再访问 SRAM；如果要查找 NFA 状态，则可直接访问 SRAM。

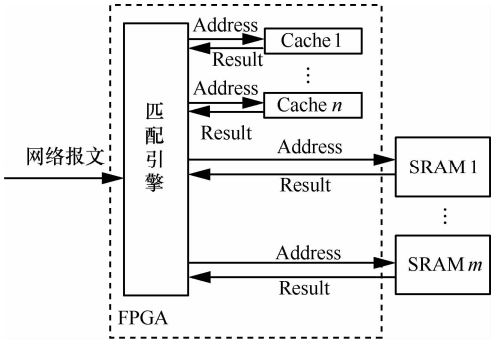


图 4　状态表的存储结构

通常这些存储在一级存储器中状态的标识并不是连续的，如果要查找某个状态的表项，就需要对所有的标识号逐一对比，提高了寻址的复杂度。最简单的解决方法就是对 DFA 中的状态进行重新编号，假设有 m_1 个状态可以存储在一级存储器中，则可以将这 m_1 个状态的标识号与原标识号为 0~m_1-1 的状态的标识号互换，这就是状态重命名。重命名后就可以直接用初始地址加偏移量的方法查找到指定状态的表项。另外，对于不在 Cache 中的状态表项可直接访问 SRAM，不需要等 Cache 未命中时再访问 SRAM。状态重命名的方法很简单，每交换一对状态的标识，只需要完整扫描一次状态表，将状态表中这 2 个状态的标识互换。在最后一级存储器中存储 HFA 的整个状态表，包括完整的 DFA 状态表和完整的 NFA 状态表。因为最后一级存储器的容量相对较大，对一级存储器中状态表项的重复存储带来的开销可以忽略。另外，存储整个状态表可以使最后一级存储器的寻址相对简单。

5　实验及分析

为评价两级存储的正则表达式匹配性能，在台式机上进行了仿真实验，性能指标主要是高速存储器的访存命中率及吞吐量。

规则集采用 L7-filter[2]的 114 条规则，生成 Becchi 在文献[12]中提出的 HFA。生成的 HFA 包含 240 453 个头部 DFA 状态、77 032 个边界状态、1 284 个尾部 NFA 状态。其中，头部 DFA 中有 1 个过渡状态集和 114 个闭集合，过渡状态集有 232 484 个 DFA 状态，每个闭集合平均有 69 个 DFA 状态。然后对头部 DFA 求解转移概率矩阵和稳态向量，得到头部 DFA 中闭集状态的访问概率排序。

测试数据集有 2 组，第一组是 1999 年 DARPA 的第一周周一的外网数据分组（323 M，共 1 362 869

个报文），由于 DAPRA 数据量小，报文较短，且缺少很多新的报文协议，对性能评价有一定的影响。第二组数据是 2012 年 11 月 15 日 21:00，在华南某 OC192 链路上被动获取的报文 CAPTURE（2.1 G，共 4 893 074 个报文）。

实验中各级存储器件的访存时间按主流存储器的频率计算，如表 2 所示。其中，FPGA 型号 Altera Stratix II EP2S180，频率为 500 MHz，综合后频率按 400 MHz 计算，访存周期为 1/2（双端口）。SRAM 型号 K7N641845M，频率 250 MHz，访存时间为 3 个周期。

表 2　　存储器件性能

存储器	频率/MHz	访存周期	访存时间/ns	吞吐量/(Gbit·s^{-1})
FPGA	400	1/2	1.25	6.4
SRAM	250	3	12	0.67

FPGA 有 9 块片上 M-RAM，每块 512 KB，可存储 100 个 DFA 状态表项。每块 SRAM 存储容量为 4 M×18 bit，存储整个 HFA 状态表项需要 15 块 SRAM。

首先讨论实验中 DFA 状态访问概率问题。理论上过渡状态集中的 DFA 状态访问概率是 0，因此只考虑闭集合中的 DFA 状态。闭集合中 DFA 状态总数为 7 969，为了方便表示，取其中概率较高的 606 个 DFA 状态。DFA 状态理论上访问概率统计如图 5 所示。考虑到不同状态间概率差异太大，为表示方便，图 5 中纵坐标取概率的常用对数。

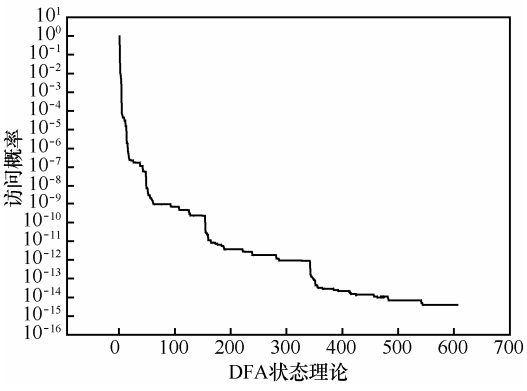


图 5　部分 DFA 状态理论访问概率

由图 5 可以看出，理论上 DFA 状态的概率分布极其集中，极个别状态理论访问概率很大，而绝大多数的状态理论访问概率很小。如状态 0 理论访问概率为 97%，而状态 606 的理论访问概率为 4×10^{-5}。

对 DARPA 报文和 CAPTURE 报文分别进行匹配实验，统计得到 DFA 状态实际访问的概率函数如图 6 所示。由图 6 可以看出，实际 DFA 状态访问概率函数与理论访问概率函数曲线走势相同，并且 CAPTURE 报文的概率函数曲线更接近于理论值。虽然实际值与理论值之间存在一定的偏差，但偏差比较小，说明通过马尔可夫链理论计算出的访问概率准确性很高。

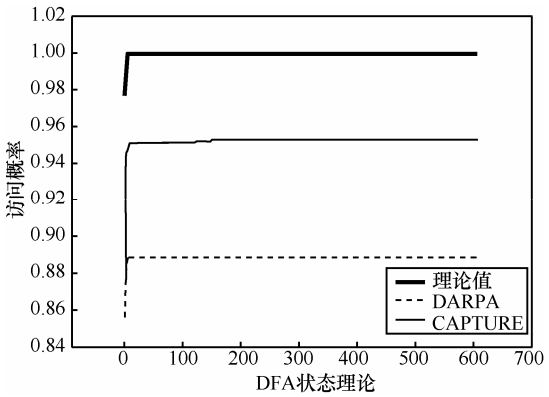


图 6　DFA 状态的访问概率函数

由图 5 和图 6 可以看出，理论上和实际上对 DFA 状态的访问都很集中，极少数状态有很高的访问概率。由此统计出实验中 DFA 命中率随一级存储器空间大小的变化，如图 7 所示。

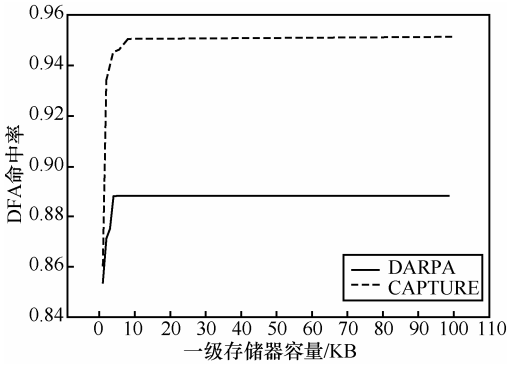


图 7　DFA 命中率与一级存储器容量关系

由图 7 可以看出，在一级存储器容量比较小时，命中率随容量的增加而急剧提高；当容量达到 15 KB 时，随着容量的增加，命中率几乎保持不变。说明一级存储器只需要很小的容量就能使 DFA 的访问达到很高的命中率。一般 FPGA 片上存储器单片容量可以达到数百千字节，事实上只需要存储数十个访问概率比较高的 DFA 状态表项，剩余的空间可以用于报文 FIFO、流管理等。另外，匹配

表 3　　FPGA+SRAM 两级存储的匹配性能

测试数据集	DFA 访问次数	NFA 访问次数	FPGA 命中次数	SRAM 访问次数	FPGA 命中率	匹配速度/(ns·byte^{-1})	吞吐量/(Gbit·s^{-1})
DARPA99	228 370 907	9 020 615	202 893 878	34 497 644	85.46%	2.81	25.60
CAPTURE	2 057 804 459	84 666 639	1 960 267 851	182 203 247	91.49%	2.16	33.26

DARPA 报文时，DFA 命中率明显略低于匹配 CAPTURE 报文，这是因为 DARPA 报文数据量小、报文短，因此对性能评估有一定影响。在实际网络环境中报文的长度对性能有一定的影响，因为报文的长度比较大时，概率的分布更接近于稳态，从而使得匹配时对状态的访问概率更接近于基于马尔科夫链理论分析得到的概率分布。

下面从 HFA 的整体访存来分析匹配性能。若使用 SRAM 单级存储的架构，SRAM 的访存性能即引擎的匹配性能，吞吐量为 0.67 Gbit/s。若采用两级存储的架构，即 FPGA 作为一级高速存储器，存储访问概率较高的 100 个 DFA 状态表项，SRAM 作为低速存储器，存储整个 HFA 的状态表项。分别对两组测试数据进行匹配，得到实验结果如表 3 所示。

从表 3 可以看出，使用 FPGA+SRAM 两级存储架构时，一级存储器 FPGA 的命中率可达到 91.49%，吞吐量可达到 33 Gbit/s，比只使用 SRAM 的性能提高 50 倍。

6　结束语

正则表达式的匹配是深度报文检测的关键技术，规则集的复杂化导致内存需求急剧增加。状态表的规模远超过高速存储器容量，大规模的状态表只能配置在低速大容量存储器中进行匹配，匹配速度严重受限于低速存储器的访存时间。网络报文的流动性又导致以 CPU 为中心的匹配引擎的 Cache 命中率不高。本文提出了一种基于 FPGA 的分级存储的正则表达式匹配引擎，使用马尔可夫链的理论求出自动机中访问概率较高的状态表项，并将它们配置在高速存储器中。实验证明，使用两级存储的架构吞吐量可达到 33 Gbit/s，比使用单级 SRAM 存储所有状态表项的方法提高 50 倍。后续研究将改进子集构造算法中的数据结构，提高子集构造算法的效率，以实现片上计算。继续研究马尔可夫链理论在混合自动机上的应用，进一步提高高速存储器的命中率。寻找合理的 NFA 结构，以实现对 NFA 状态表项的快速访问。结合使用高效的压缩算法，减少自动机对二级存储器的内存需求。最后实现基于两级存储的高速匹配引擎。

参考文献：

[1] 张树壮, 罗浩, 方滨兴. 面向网络安全的正则表达式匹配技术[J]. 软件学报, 2011,22(8):1838-1854.
ZHANG S Z, LUO H, FANG B X. Regular expressions matching for network security[J]. Journal of Software, 2011,22(8):1838-1854.

[2] Application layer packet classifier for linux[EB/OL]. http://l7- filter.sourceforge.net.

[3] Snort user manual, the snort project[EB/OL]. http://www.snort.org/Assets/82/snort_manual.pdf.

[4] Introduction to bro[EB/OL]. http://www.bro-ids.org/wiki/index.php/Bro.

[5] YU F, CHEN Z, DIAO Y, *et al*. Fast and memory-efficient regular expression matching for deep packet inspection[A]. Architecture for Networking and Communications Systems, ANCS2006 ACM/IEEE Symposium on[C]. 2006.93-102.

[6] BECCHIMCADAMBI S. Memory-efficient regular expression search using state merging[A]. INFOCOM 2007, 26th IEEE International Conference on Computer Communications IEEE[C]. 2007.1064-1072.

[7] 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[J]. 计算机研究与发展，2008,45(8): 1299-1306.
CHEN S H,SU J S, FANH P, *et a*l. An FSM state table compressing method based on deep packet inspection[J]. Journal of Computer Research and Development,2008, 45(8):1299-1306.

[8] KUMAR S, DHARMAPURIKAR S, YU F, *et al*. Algorithms to accelerate multiple regular expressions matching for deep packet inspection[J]. 2006 ACM SIGCOMM Computer Communication Review, 2006,36(4): 339-350.

[9] 杨毅夫，刘燕兵，刘萍. 正则表达式的 DFA 压缩算法[J].通信学报，2009,30(10) :36-42.
YANG Y F, LIU Y B, LIU P. Effective algorithm of compressing regular expressions DFA[J]. Journal on Communications, 2009,30(10): 36-42.

[10] SOMMER R, PAXSON V. Enhancing byte-level network instruction detection signatures with context[A]. ACM Conf on Computer and Communication Security[C]. 2003.262-271.

[11] KUMARS, CHANDRASEKARAN B, TRUNER J, *et al*. Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia[J]. Proceedings of ACM/IEEE ANCS[C]. 2007.155-164.

[12] BECCHI M, CRWOLEY P. A hybrid finite automaton for practical deep packet inspection[A].ACM Computer Communication Networks ,CoNEXT 2007[C]. New York, NY, USA,2007.

[13] LIN C H, LIU J C. M-DFA(multithreaded DFA): an algorithm for reduction of state transitions and acceleration of REGEXP matching[A]. Proceedings of the 8th ACM/IEEE Symposium on Architectures for Networking and Communications Systems[C]. 2012.79-80.

[14] 王磊，陈曙晖，苏金树等. 深度报文检测中基于 GPU 的正则表达式匹配引擎[J].计算机应用研究，2010, 27(11): 4324-4327.

（下转第 63 页）