

文章编号:1001-9081(2007)09-2184-03

基于正则表达式的深度包检测算法

丁 晶,陈晓岚,吴 萍

(华东师范大学 计算中心,上海 200062)

(51051211003@student.ecnu.edu.cn)

摘 要:在深入分析了 DFA 状态数对算法性能影响的基础上,提出了一种新的基于正则表达式的深度包检测算法,该算法保证在任意有限的系统资源下算法的时间复杂度空间复杂度最小。在 Linux 下实现了该算法,并对基于 L7-filter 模式集合的网络数据包进行了大量检测实验。结果表明,与已有的正则表达式算法比较,该算法的时间复杂度和空降复杂度最小。

关键词:深度包检测;模式匹配;正则表达式;确定性有限自动机

中图分类号: TP393.08 **文献标志码:** A

Deep packet inspection algorithm based on regular expressions

DING Jing, CHEN Xiao-lan, WU Ping

(Computer Center, East China Normal University, Shanghai 200062, China)

Abstract: This paper proposed a new DFA-based pattern matching algorithm. Based on the analysis of the impact of the number of DFA states on the algorithm performance, further improvement to the algorithm was made by introducing a DFA state number optimization algorithm. The proposed algorithm has been implemented in Linux environment and lots of experiments have been done. Experimental results show that the performance of the proposed algorithm is much better than others.

Key words: deep packet inspection; pattern matching; regular expressions; Deterministic Finite Automation(DFA)

0 引言

随着专门针对应用层攻击现象的增多,传统的状态检测防火墙有效性越来越低。防火墙的功能重心从网络层发展到了应用层,因而诞生了深度包检测技术。深度包检测技术不仅检测网络层和传输层数据包头部,而且深入到应用层数据包的有效载荷所封装的内容中,搜寻合法或非法的内容以决定是否允许数据包通过。

在深度包检测技术发展的同时,传统上用于过滤数据包内容的模式集合(包含模式的匹配串)逐渐被正则表达式集合所代替。虽然正则表达式具有灵活、高效的特点,在模式匹配时比字符串表现得更优异,然而在网络应用中,一个典型的模式集合往往由上百个正则表达式和数以万计的状态数组成。为了达到最佳的匹配速度,将模式集合构造成一个有限自动机,所需的内存可达几百 M,甚至几个 G,这就极大地影响了基于正则表达式检测算法的效率,从而使得时空复杂度都比较大。这样的速度和性能在现实应用中是很难被接受的。针对这种情况,本文提出了一个新的基于正则表达式深度包检测算法,并且在 Linux 下面进行了验证了,得到了有效的实验数据。通过数据结果进行比较,相对于传统的基于正则表达式的匹配算法,本文提出的算法在时间复杂度和空间复杂度方面都得到了一定程度的提高。

1 深度包检测原理与正则表达式

深度包检测技术使应用程序通信管理设备能够深入分析 TCP 或 UDP 通信流量的内容。当 IP 数据包、TCP 数据流或

UDP 包经过管理设备时,将其重新组合,从而得到整个应用程序的内容,然后按照企业定义的策略对应用程序进行操作。

信息被分割成小的数据包,以便能够快速通过网络。负载均衡设备在这些小数据包的传送途中截获它们,然后应用程序通信管理设备使用深度包检测技术深入分析 TCP 或 UDP 通信流量的内容。当 IP 数据包、TCP 数据流或 UDP 包经过管理设备时,将其重新组装为原始的数据,并将这些数据缓存。通过扮演特定应用程序数据流代理的角色,通信管理设备继续获取相关信息,更多的内容被检测到,同时寻找已经定义的变量,根据这些变量决定采取的动作。用户可以使用一定的规则或策略来定义这些变量,使这些策略基于应用程序的类型或者数据源和最终目标。一旦通信管理设备定位了有效载荷的信息,它就会向能够最好处理客户请求的应用程序或资源发送数据。深度包检测同样可以应用于检测应用程序或服务中的变量的正确性。如果这些变量不存在,那么请求就会被丢弃,同时将事件记录到日志文件并向管理员发送警报。

正则表达式描述了一种字符串匹配的模式,可以用来检查一个串是否含有某种子串、将匹配的子串做替换或者从某个串中取出符合某个条件的子串等。确定有限状态自动机是由下述 5 个元素构成的数学模型:

$$(S, \Sigma, \delta, S_0, F)$$

其中:

S 是有限的,非空的状态集;

Σ 是输入字母表;

收稿日期:2007-03-16;修回日期:2007-06-01。

作者简介:丁晶(1979-),男,上海人,硕士研究生,主要研究方向:现代软件工程;陈晓岚,女,上海人,硕士研究生,主要研究方向:多媒体;吴萍,女,上海人,副教授,主要研究方向:现代软件工程。

δ 是转移函数。 $\Sigma\delta(s, a) = S'$ 意味着:当现行状态为 s 、输入字符为 a 时,将换到下一个状态 S' 。我们称 S' 为 s 的后继状态。

$S_0 \in S$ 是初始状态;

$F \subseteq S$ 是非空的终结状态集。

在深度包检测技术中,正则表达式得到了充分的应用。可以人为地将所需检测的内容表达成一个个的模式,每一个模式我们可以用正则表达式去进行描述,从而处理成计算机能够识别的一种语言。在处理正则表达式的时候,将正则表达式转化成 DFA 来进行处理。

2 存在的问题

对于给定的正则表达式集合,将集合中的所有正则表达式构造成一个 DFA,可以达到最快的匹配速度。然而一般而言,多个模式对应一个 DFA 的状态数要远远大于一个模式对应一个 DFA 状态数的总和。因为对任一正则表达式,都能够用一个状态有限的 DFA 来表示,DFA 占据的内存量的大小,取决于状态的数量和每个状态的转换的数量的乘积(为了表述方便,下面内存的大小按照状态数来衡量)。因此上述表述也可以理解为:多个模式对应一个 DFA 所需内存要远远大于一个模式对应一个 DFA 所需内存的总和。

在 Linux L7-filter 作为匹配数据包的模式集中,每个模式对应一个 DFA 的状态数总和与多个模式对应一个 DFA 的状态数的比较如图 1 所示。

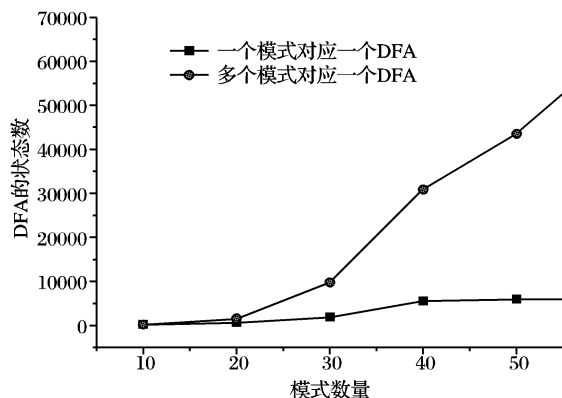


图1 模式状态数量图

由图1可以看出当模式数量在10~20时,一个模式对应一个 DFA 的状态数总和与多个模式对应一个 DFA 的状态数相差不大。但当模式数量多于20时,将多个模式构造成一个 DFA 的状态数迅速增加。对于 m 个模式集合,若每个模式包含一个通配符,则需要 $O(2^m)$ 个状态数,那么每增加一个包含一个通配符的模式,DFA 的状态数就要增加一倍。 m 个模式中若每个模式包含 x 个通配符,就需要 $(x+1)^m$ 个状态。因为作为数据包匹配的每条模式中通常都包含了大量的通配符,所以将多个模式构造成一个 DFA 时,势必会引起状态数的指数性增加。虽然将多个模式构造成一个 DFA 可以提高匹配数据包的速度,但是会同时引起内存指数性的增长,当模式集合达到一定规模时,就会超出系统的物理内存,此时系统使用虚拟内存作为补充,结果导致了匹配数据包速度极大地降低。这就是目前的基于正则表达式的深度包检测技术所存在的问题。针对该问题,本文提出了一种构造最优 DFA 状态数的算法,该算法保证在任意有限的系统资源下时间复杂度最小。

3 算法描述

算法大体分为3个部分:1) 根据计算 $\{P_1P_2\cdots P_m\}$ 中每个模式生成 DFA 的状态数与模式集合中两两构造 DFA 的状态数生成图;2) 根据图构造 $\{M_1M_2\cdots M_k\}$;3) 判断是否 m 个模式都已经加入 $\{M_1M_2\cdots M_k\}$ 中,如果是,算法结束,如果不是,按照另外的方式将剩余的模式加入 $\{M_1M_2\cdots M_k\}$ 。具体如下所示:

1) 构造图

输入集合 $\{P_1P_2\cdots P_m\}$ 中的每个模式 P_i ,通过词法分析器 Flex 可以得到给定模式生成 DFA 的状态数 D_{P_i} ,将模式集合 $\{P_1P_2\cdots P_m\}$ 两两作为一组,即对任意的 P_iP_j 构造 DFA,计算其状态数 $D_{P_{ij}}$,将 D_{P_i}, D_{P_j} 保存在数组 $s[m][m]$ 中,其中 D_{P_i} 存放在 $s[i][i]$ 中, $D_{P_{ij}}$ 保存在 $s[i][j]$ 中($s[j][i]$ 与 $s[i][j]$ 的值相同)。

根据若两个模式构造的 DFA 状态数大于其分别构造 DFA 状态数之和,那么两个模式,即图中的两点之间存在一条边。即计算 $s[i][j]$ 是否大于 $s[i][i]$ 与 $s[j][j]$ 之和,若成立,那么图 $Graph[i][j] = 1$,同时 $Graph[j][i] = 1$;若不成立, $Graph[i][j] = 0, Graph[j][i] = 0$ 。其中 m 个模式对应图中的 m 个顶点。

2) 构造 $\{M_1M_2\cdots M_k\}$

第1步:开始构造集合 $\{M_1M_2\cdots M_k\}$ 。根据图 Graph,查找图中度数最小的点 t ,加入 $M_i, tmpM_i$ (为了计算方便,保存中间状态的集合)中,作为模式集合 M_i 中的一员,并在图中去掉点 t ,同时将可用内存变更为 $ME = ME - D_{P_t}$ 。

第2步:在图中剩余的点里查找和当前 M_i 中的点连接最少的点 j 加入 $tmpM_i$,比较 $D_{tmpM_i} > \sum_{i \in M_i} D_{P_i} + ME$,即需要比较将模式 j 加入到模式集合 R_i 中是否会引起内存的迅速增加,若成立那么 M_i 构造结束,进行第3步;若不成立,那么将点 j ,加入到 $M_i, tmpM_i$ 中,并在图中去掉点 j ,同时将可用内存变更为 $ME = ME - D_{R_i} + D_{P_j}$,然后返回第2步;若此时无点 j 的存在,转到第3步。

第3步:计算图中剩余点的度数和,若度数和 > 1 即图中存在边,那么返回第2步,对 M_{i+1} 进行构造;若度数之和等于0,即剩余的点为孤立点,或图中已无剩余点,那么进行第3部分的计算。

3) 对构造的模式集合 $\{M_1M_2\cdots M_k\}$ 进行最后的处理

判断图中是否已无点剩余,若是,则 $\{M_1M_2\cdots M_k\}$ 构造结束;若否,对图中剩余的点进行新的构造。首先判断剩余的点即剩余的模式构造成一个 DFA 是否超过当前内存的限制,若否,那么将剩余的模式构造成一个 DFA,完成 $\{M_1M_2\cdots M_k\}$ 的计算;若是,则将剩余点(即模式)中的一个单独构造一个 DFA, $M_s, s < k$,同时 $ME = ME - D_{M_s}$,再次比较将剩余模式构造成一个 DFA 是否超过当前内存的限制,若是,再将剩余模式单独划分出一个,再去判断剩余的 mode 集,直到 m 个模式均已加入到集合 $\{M_1M_2\cdots M_k\}$ 中。

4 实验及性能分析

实验环境:CPU 1.6 GHz,内存 765 MB, Redhat 9.0 操作系统。采用 Linux L7-filter 的 http 规则来进行匹配,以判断本算法在性能上的提高。

实验将抓取的 http 数据包作为需要匹配的数据包,判断是否允许该数据包通过(模式集合 $\{P_1P_2\cdots P_m\}$ 中包含了 http 模式),匹配方式有 3 种,第 3 种匹配方式是本算法的匹配方式:

方式一: m 个模式组成 m 个 DFA;

方式二: m 个模式组合成 k 个 DFA;

方式三: m 个模式组合成一个 DFA。

本实验分成 2 种情况对 3 种方式进行比较:1)在内存一定的前提下模式集合不断的增大;2)在 m 个模式集合一定的条件下,内存不断增大(状态数不断扩大)。

注:为了保证结果的精确性,本实验的数据是运行 10 000 次的取得平均值,其中为了表示的方便,时间是运行 100 次匹配数据包时间的总和。

1)内存(30 000 个状态数)一定,模式集合(从 $m = 10$ 到 $m = 60$)依次增加。

(1)空间复杂度比较(以 DFA 状态数来衡量)

表 1 内存一定的空间复杂度比较

方式	模式数					
	10	20	30	40	50	60
m 个 DFA	274	694	1910	5 598	6 012	6 194
k 个 DFA	249	1 613	9 831	5 884	6 340	6 562
一个 DFA	249	1 613	9 831	30 944	43 613	610 808

如表 1 所示,当 $m \leq 30$ 时, $D_m < 30\,000$, 所以将 m 个模式组成 k 个 DFA 的状态数与组成一个 DFA 的状态数相等,即当 $m \leq 30$ 时, $k = 1$, 本算法将 30 个模式构造成一个 DFA,以达到最佳的时间复杂度;在 $m > 30$ 时,由于模式中通配符的增加, m 个模式组成一个 DFA 状态数成指数性的增长,而将 m 个模式组成 k 个 DFA 的状态数与方式一(m 个模式构造 m 个 DFA)图中最小的状态数基本一致,因此本算法方式二的空间复杂度明显优于方式三的空间复杂度。综上所述,方式一的空间复杂度 < 方式二的空间复杂度 < 方式三的空间复杂度。

(2)时间复杂度比较

表 2 内存一定的时间复杂度比较

方式	模式数					
	10	20	30	40	50	60
m 个 DFA	1.18	2.31	3.38	4.49	5.64	6.73
k 个 DFA	0.18	0.16	0.16	0.49	0.60	0.67
一个 DFA	0.18	0.16	0.16	—	—	—

如表 2 所示,采用 m 个模式构造 m 个 DFA 的方式,因为需要进行多次匹配操作,所用时间成线性增长的趋势;将 m 个模式构造一个 DFA,由空间复杂度分析可知,当 $m > 30$ 时,构造一个 DFA 超出了内存的限制,此时系统会使用虚拟内存

表 3 内存不定的空间复杂度比较

方式	模式数							
	7 000	10 000	20 000	30 000	40 000	50 000	55 000	60 000
m 个 DFA	6 194	6 194	6 194	6 194	6 194	6 194	6 194	6 194
k 个 DFA	6 107	6 493	6 562	6 562	6 562	6 562	16 636	14 545
一个 DFA	61 808	61 808	61 808	61 808	61 808	61 808	61 808	61 808

表 4 内存不定的时间复杂度比较

方式	模式数							
	7 000	10 000	20 000	30 000	40 000	50 000	55 000	60 000
m 个 DFA	6.73	6.73	6.73	6.73	6.73	6.73	6.73	6.73
k 个 DFA	1.53	0.76	0.63	0.62	0.61	0.61	0.28	0.28
一个 DFA	—	—	—	—	—	—	—	—

作为补充,但是虚拟内存是操作系统在硬盘上开辟的一块磁盘空间,使用虚拟内存意味着需要增加读写硬盘的时间,结果导致了匹配时间的迅速增加;本算法采用 m 个模式构造 k 个 DFA,由于空间复杂度较小,随着模式的增多,不会超出内存的限制,同时因为进行匹配的次数也优于方式一,因此可以达到最小的时间复杂度。

2) m 个模式集合($m = 60$)一定的条件下,内存不断增大(从 $ME = 10\,000$ 到 $ME = 60\,000$)。

(1)空间复杂度比较(以 DFA 状态数来衡量)

如表 3 所示,将 m 个模式构造成 m 个 DFA 的状态数之和为 6 194,而 m 个模式集合对应一个 DFA 的状态数为 61 808,几乎是前者的 10 倍,而且已经超出了实验最大值 60 000,所以需要采用本算法将 m 个 DFA 构造成 k 个 DFA, DFA 的状态数随着内存的增加变化不是很大,基本与 m 个 DFA 的总状态数近似,只是当内存达到 54 500 左右时,因为内存的增大,更多模式可以对应一个 DFA,所以引起了状态数的增加,但是仍远远小于 61 808。综上所述,本算法在模式集合一定,内存不断增大的情况下具有较好的空间复杂度。

(2)时间复杂度比较

由空间复杂度分析可得,将 m 个模式构造成一个 DFA 状态数已经超出了内存的最大限制,因此也用到了虚拟内存,结果导致了匹配时间的迅速增加。由表 4 可知,方式二的时间复杂度 < 方式一的时间复杂度 < 方式三的时间复杂度,即本算法的时间复杂度最优。

综合实验的两个情况,无论是模式的增加还是内存的变化,由实验数据可知,本算法达到了最小的时间复杂度。

5 结语

根据前面的分析得出:当正则表达式用于模式匹配时,随着模式集合的增长,因为状态数的不断扩大,引起内存的占有量不断增加,尤其当模式中包含大量通配符时,内存的占有量会成指数性增长,因为用于匹配数据包的模式通常都包含了大量的通配符,所以当正则表达式用于深度包检测时,内存的问题成为最需解决的问题。针对此问题,提出了一种当用正则表达式模式集合进行检测时在保证一定匹配时间的前提下,减少内存的算法。并在 Linux 环境下,通过 Linux L7-filter 模式集匹配网络中抓取得 http 数据包进行了实验。通过两个方面验证内存的占有量的情况:1)在内存一定的条件下,模式个数不断增加;2)在模式集合一定的条件下,内存不断增加。由实验数据可得,不论在那种情况下,本算法都实现了在

保证一定匹配时间的前提下,内存的占有量得到了明显减少的目的,因此证明本算法是合理有效的。

今后,还可以从 DFA 的压缩技术方面进一步研究,找到更加节约内存的目标。同时随着网络传输速度的快速提高,数据包的检查要求以极高的速度分析、检测及重新组装应用流量,以避免给应用带来时延。单纯依靠软件方法可能还不足以提供快速的应用层数据分析,今后的工作方向也可以考虑采用基于硬件方法实现更加快速的深度包检索引擎。

(下转第 2193 页)

1) 当 A 、 B 均为诚实主体时, $G1$ 、 $G2$ 、 $G3$ 均满足, 即 ZG 协议满足非否认性和公平性。

2) 当 A 为恶意主体时, $G1$ 、 $G2$ 满足, 说明协议满足非否认性。而 $G3$ 不满足, 说明协议不满足公平性。通过查看两个子阶段的公平性结果发现: $Fairness_1$ 满足, 即 A 、 B 分别拥有了 EOR 和 EOO ; $Fairness_2$ 不满足, 由于 A 不可能做出对自己不利的事情, 因此 A 得到了 con_K , 而 B 没有得到, A 在协议中占了优势。故, 该协议对 B 不公平。

3) 当 B 为恶意主体时, ZG 协议的非否认性和公平性均满足。

这个分析结果与文献[15]得到的结果一致, 表明了我们提出的模型的有效性和正确性。

5 结语

本文针对非否认协议提出了一个专门用于分析非否认性和公平性的一阶逻辑模型:

1) 引入了谓词 $agent(m, p(X, flag))$, 将消息和消息来源绑定在一起, 为解决非否认性和公平性问题奠定了基础。引入了 $match(a, b)$ 谓词描述对消息的检验和仲裁过程, 使协议刻画更加精确。

2) 在参与方和仲裁的描述中添加了对应性事件 $askJSays$ 和 $answerJSays$ 解决非否认性问题, 更接近于非否认协议实际。文献[13]将非否认性看作了认证问题来解决, 通过参与方之间的相互认证来判断非否认性是否满足, 这种方法不太精确, 有时会漏掉一些攻击。

3) 对公平性的定义充分考虑了协议运行中的各个阶段并形式化了“优势”的概念, 相比较于那些只考虑协议完成之后的状态更加精确, 更接近公平性的本质。

目前还没有专门针对非否认协议的一阶逻辑模型。通过具体的协议验证实例可以看出, 本文提出的模型是有效的; 同时, 通过比较还可以看出该模型具有针对性强、更加精确、更加接近于非否认协议实际的优点。下一步的工作将对本文提出的方法进行完善和扩展, 使其能够在安全特性不满足时构造出攻击路径, 并可以对带时间戳的非否认协议进行分析。

参考文献:

- [1] ZHOU J Y, GOLLMANN D. A fair non-repudiation protocol [C]// Proceedings of the 1996 IEEE Symposium on Security and Privacy. [S. l.]: IEEE Press, 1996: 55 – 61.
- [2] ASOKAN N. Fairness in Electronic Commerce [D]. Waterloo: University of Waterloo, 1998.
- [3] ASOKAN N, SHOUP V, WAIDNER M. Asynchronous protocols for optimistic fair exchange [C]// Proceedings of IEEE Symposium on Research in Security and Privacy. [S. l.]: IEEE Press, 1998: 86 – 99.
- [4] GARAY J A, JAKOBSSON M, MACKENZIE P. Abuse-free optimistic contract signing [C]// Proceedings of Advances in Cryptology (Crypto '99). Berlin: Springer-Verlag, 1999: 449 – 466.
- [5] BLANCHET B. An efficient cryptographic protocol verifier based on Prolog rules [C]// 14th IEEE Computer Security Foundations Workshop. [S. l.]: IEEE Press, 2001: 82 – 96.
- [6] SCHNEIDER S. Formal analysis of a non-repudiation protocol [C]// Proceedings of the 11th IEEE Computer Security Foundations Workshop. [S. l.]: IEEE Press, 1998: 54 – 65.
- [7] KREMER S, RASKIN J F. A game-based verification of non-repudiation and fair exchange protocols [C]// LARSEN KG, NIELSEN M, ed. 12th International Conference on Concurrency Theory, CONCUR. LNCS 2154. Berlin: Springer-Verlag, 2001: 551 – 565.
- [8] 蓝荣胜, 陈大伟, 郭云川, 等. 公平非否认协议的有限状态分析 [J]. 计算机科学, 2005, 32(8): 83 – 86.
- [9] ZHOU J Y, GOLLMANN D. Towards verification of non-repudiation protocols [C]// Proceedings of the 1998 International Refinement Workshop and Formal Methods Pacific. Berlin: Springer-Verlag, 1998: 370 – 380.
- [10] 范红, 冯登国. 一个非否认协议 ZG 的形式化分析 [J]. 电子学报, 2005, 33(1): 171 – 173.
- [11] 黎波涛, 罗军舟. Zhou-Gollmann 不可否认协议的一种新的改进 [J]. 计算机学报, 2005, 28(1): 35 – 45.
- [12] BELLA G, PAULSON L. Mechanical proofs about an on-repudiation protocol [C]// Proceedings of 14th International Conference on Theorem Proving in Higher Order Logic. Berlin: Springer-Verlag, 2001: 91 – 104.
- [13] Judson Santiago. Study for Automatically Analysing Non-repudiation [EB/OL]. [2006 – 10 – 10]. <http://www.avispa-project.org/papers/SantiagoV-CRISIS05.pdf>.
- [14] 卿斯汉, 李改成. 公平交换协议的一个形式化模型 [J]. 中国科学, 2005, 35(2): 161 – 172.
- [15] GURGINS S, RUDOLPH C. Security analysis of (un-) fair non-repudiation protocols [C]// Formal Aspects of Security (FASec'02). Berlin: Springer-Verlag, 2002: 97 – 114.
- [16] ABADI M, BLANCHET B. Computer-assisted verification of a protocol for certified email [C]// The 10th Int'1 Symposium (SAS'03), LNCS 2694. Berlin: Springer-Verlag, 2003.
- [17] 范钰丹, 韩继红, 王亚弟, 等. 非否认协议形式化分析技术 [J]. 计算机应用, 2006, 26(11): 2610 – 2614.

(上接第 2186 页)

参考文献:

- [1] 柳岸, 龙雅琴, 古乐野. 基于包过滤技术的网络安全的研究 [J]. 计算机应用, 2006, 26(9): 2160 – 2161.
- [2] 刘更楼, 丁常福, 姜建国. 基于状态检测的防火墙系统研究 [J]. 航空计算技术, 2004, 34(1): 122 – 125.
- [3] 王栋. 防火墙深度包检测技术研究 [M]. 西安: 西安电子科技大学, 2005.
- [4] DENNING D. An intrusion detection model [J]. IEEE Transactions on Software Engineering, 1987, SE-13(2): 222 – 223.
- [5] COMMENTZ-WALTER B. A string matching algorithm fast on the average [C]// Proceedings of the 6th International Colloquium on Automata, Language and Programming. LNCS 71. Berlin: Springer-Verlag, 1979: 118 – 132.
- [6] WU S, MANBER U. A fast algorithm for multi-pattern searching, TR-94-17 [R]. Arizona: University of Arizona, 1994.
- [7] TUCK N, SHERWOOD T, CALDER B, et al. Deterministic memory-efficient string matching algorithms for intrusion detection [C]// Proceedings of IEEE Infocom. [S. l.]: IEEE Press, 2004: 333 – 340.
- [8] THOMPSON K. Programming techniques: regular expression search algorithm [J]. Communications of the ACM, 1968, 11(6): 419 – 422.
- [9] 卢开澄. 计算机算法导引 [M]. 北京: 清华大学出版社, 1996.
- [10] LEVANDOSKI J, SOMMER E, STRAIT M. Application layer packet classifier for Linux [EB/OL]. [2006 – 12 – 10]. <http://l7-filter.sourceforge.net/>.
- [11] PAXSON V. Flex: A Fast Scanner Generator: version 2.5 [EB/OL]. [1998 – 12 – 10]. http://www.gnu.org/software/flex/manual/html_mono/flex.html.