Snort 中 BM 模式匹配算法的研究与改进①

Research and Improvement of BM – algorithm for pattern matching in Snort

费洪晓　戴宏伟　肖新华　（中南大学信息科学与工程学院 长沙 410075）

摘要：首先详细的阐述了入侵检测系统 Snort 的 BM 模式匹配算法的思想，在此基础上提出了一种改进的 BM 算法，该算法在重复后缀较多的情况下，能有效的加快模式匹配的速度，提高入侵检测的效率。

关键词：模式匹配 BM 算法 入侵检测 Snort

1 引言

模式匹配是指在一个目标文本 T 中查找某个特定的子串，使得这个子串与已知的模式串 P 相等。如果在 T 中找到等于 P 的子串，则称匹配成功，否则称匹配失败。作为 IDS 的最基本的技术，也是最关键的技术，它的速度的快慢直接影响到入侵检测系统（IDS）的工作效率。著名的轻量级入侵检测系统 sonrt 采用的是 BM 模式匹配算法，该算法被称为亚线性算法，其平均匹配速度比同类型的 KMP 算法还要快 3—5 倍。本文对 BM 算法进行了改进，使之更适合与目标串中重复后缀较多的情况。

表 1

X:				B	D	H	U		
delta1(X):				1	6	2	8		
j:	0	1	2	3	4	5	6	7	8
P[j]:	$	H	D	B	H	B	H	B	H
rpr(j):	-6	-5	-4	-3	3	-1	3	0	8
delta2(j):	15	14	13	12	6	10	6	9	1

表 2

	H	D	B	U	D	B	D	B	H	B	H	B	H	D	B	H	B	H	B	H
第一次	H	D	B	H	B	H	B	H												
第二次		H	D	B	H	B	H	B	H											
第三次						H	D	B	H	B	H	B	H							
第四次														H	D	B	H	B	H	B	H

注：第一次滑动 1，第二次滑动 4，第三次滑动 7

2 BM 算法

BM 算法的基本思想是：如果从模式串的右边开始匹配，往往会比从左边开始获得更多的启发。即匹配的第一步是将目标串 T 与模式串 P 两者的左端对齐，然后从 P 的末字符开始往左对比 T 中相对应的字符。当 P 中字符 a 与 T 中对应字符 b 失配时，同时共有 3 条启发性规则指导 P 滑动（以 T 为参照）到下一个适当的位置，哪种规则下滑动的距离最大，便采用哪一种。

规则一，如果 b 没有被包含在模式串 P 中，那么 T 中从 b 开始，长度等于 strlen（P） = m 的子串是不可能和 P 匹配成功的。

规则二，如果 b 被包含在 P 中，且 b 在 a 的左边，则我们可以毫不犹豫的滑动 P 将这个 b（如果 a 的左边出现多个 b，则选最靠近 a 的 b）与 T 中的 b 对齐。万一 b 只出现在 a 的右边，那就将 P 向右滑动一个字符的位置。

规则三，假设在 a 与 b 失配之前已经匹配了一个

① 基金项目：国家自然科学基金资助（60173041），湖南省自然科学基金资助（02JJY2094），湖南省科技计划项目（2006JT1040）

长度为 i 的子串 S，如果 P 中存在其他与 S 相同的子串 S′，且 S′的前一个字符不为 a，则滑动 P 将该 S′（如果存在多个 S′，则取最靠右的 S′）与 T 中的 S 对齐。倘若 P 中并不存在 S′，而是存在 P 的某个最长前缀 S″等于 S 的一个后缀，那么就滑动 P 将这个最长前缀 S″对齐 T 中 S 的相应后缀。

功匹配 m－j 个字符后 P_j 与 T_{i+j} 失配，根据规则三的思想，如果 P 中还存在与 $P_{j+1}\cdots P_m$ 或者 $T_{i+j+1}\cdots T_{i+m}$ 一致的子串 S′，则应当把 S′与 $T_{i+j+1}\cdots T_{i+m}$ 对齐，此时函数 rpr（j）的值就是 S′的首字符在 P 中的位置序号。即便不存在 S′而只存在上面提到的最长前缀 S″，由于引入了 $ 字符的概念，把缺位用 $ 代替，计算方法也是一致

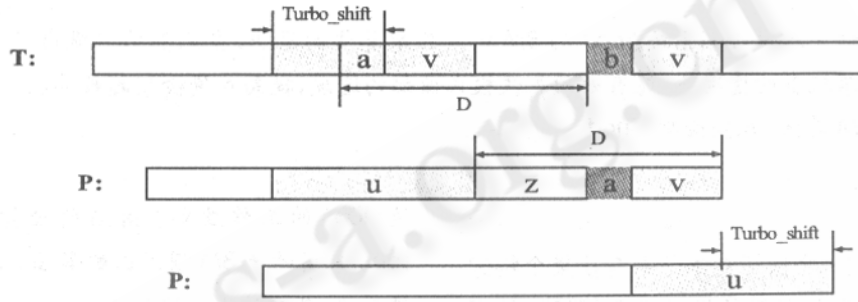

图 1　|v| < |u| 时触发 turbo－shift

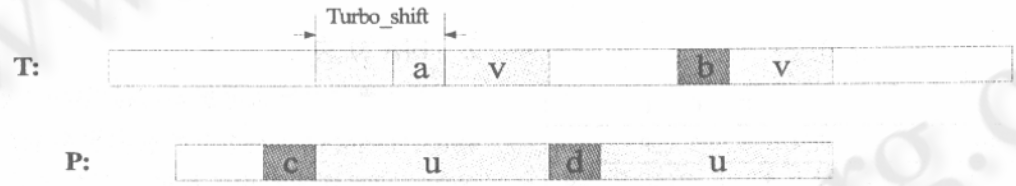

图 2

那么如何求具体的滑动值呢？算法为规则一和规则二引入滑动偏移量函数 delta1（x），其中的参数 x 为 T 中失配位置的字符。delta1（x）数学公式如下：

$$delta1(x)=\begin{cases} m & x\text{ 不在 P 中出现或 x 只出现在 P 的尾部} \\ m-j & \text{其它情况，其中 } j=\max\{j:P_j=x,1\le j\le m-1\} \end{cases}$$

根据 delta1 的值可以求得 P 的实际移动距离为 max（1，delta1（b）－i）。

同样也为规则三引入滑动偏移量函数 delta2（x），但是参数 x 并不是字符而是 P 中字符的位置序号。函数 delta2 相比 delta1 要复杂一些，所以这里再引入一个约定、一个定义和一个函数 rpr（x）。算法约定当 P_i 中的 i 小于 1 时，P_i 的值都为字符 $。同时定义，对于串 $C_1\cdots C_n$ 与 $D_1\cdots D_n$，如果 $C_i = D_i$ 或者 $C_i =$ $ 或者 $D_i =$ $（$1\le i\le n$），则说明串 $C_1\cdots C_n$ 与 $D_1\cdots D_n$ 是一致（匹配）的。对于函数 rpr（x）的意义，假设在某一趟匹配中成

的，同时也必须注意到在这种情况下，rpr（j）的值是可以小于等于 0 的。rpr（j）的数学公式如下：

$$rpr(j)=\max(k:P_{j+1}\cdots P_m=P_k\cdots P_{k+m-j-1}) \quad k\le 1 \text{ 或者 } P_{k-1}\ne P_j$$

求得 rpr（j）的值，即可知 delta2（j）= m＋1－rpr（j）。此时 P 实际的滑动量为 delta2（j）－j。假如我们要在目标串 T：HDBUDBDBHBHBHUBUBDBH 中查找模式串 P：HDBHBHBH。表 1 给出了 delta1 和 delta2 函数的值。具体匹配过程如图 2 所示。

3　改进的 BM 算法

通过对上面第三次匹配的观察，我们不难发现，第二次匹配中的已匹配后缀“BH”可以在第三次中被利用到，从而可以减少两次比较次数，这就是改进算法的出发点。

改进的算法设置了一个因子 u 来记录匹配成功

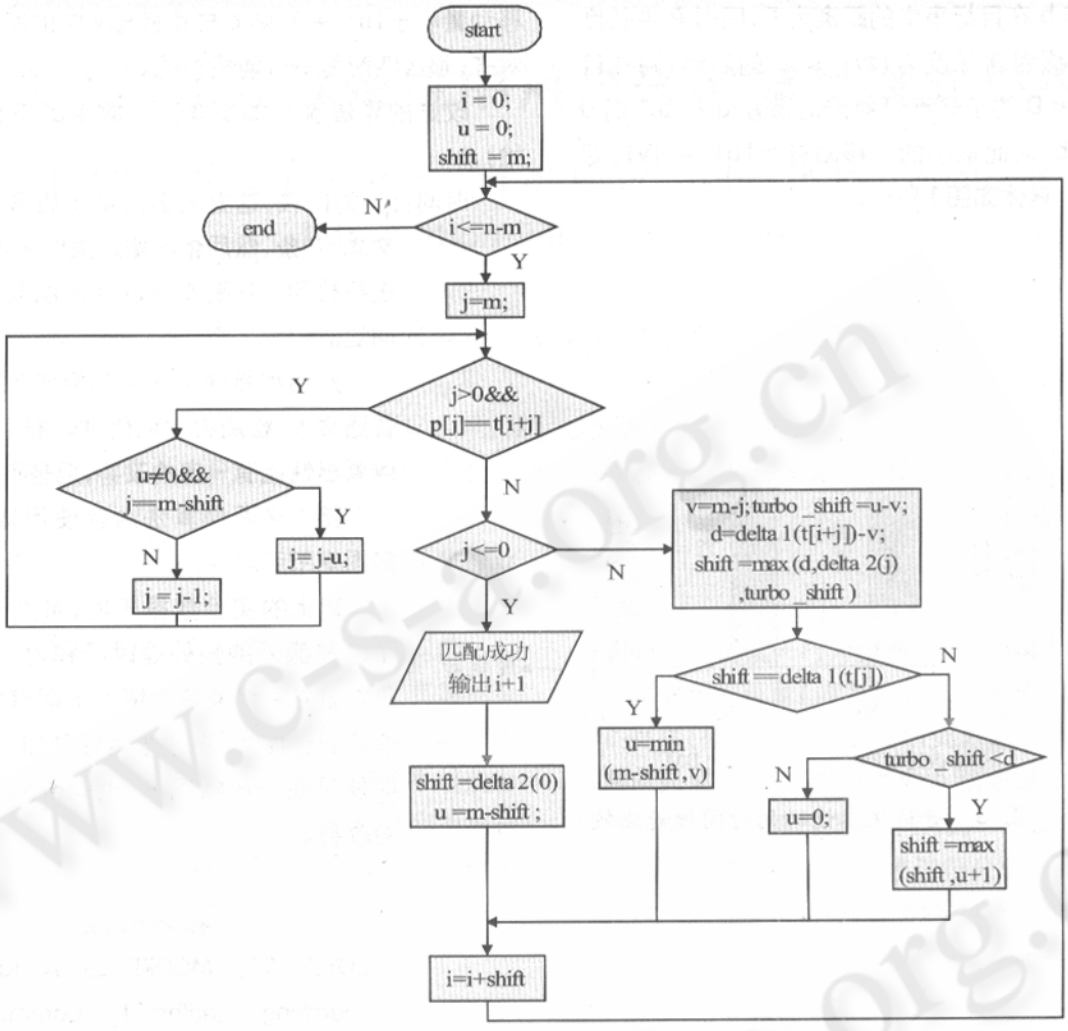

图 3　BM 改进算法与原算法的性能比较

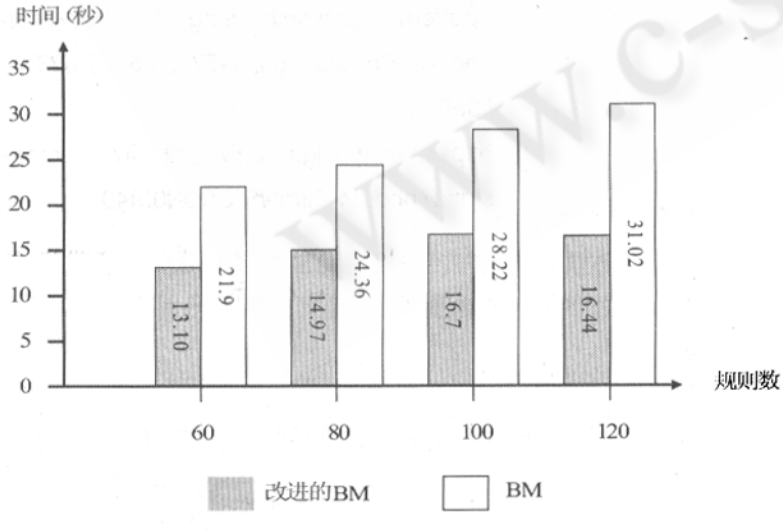

图 4　特殊文本情况的时间性能比较

的那些后缀，若上次匹配结束后是根据指导规则三来滑动 P 的，则本次匹配可以从两个方面受益：一，可以根据记录 u，有可能减少当前匹配的字符比较次数；二，如果当前的已匹配后缀长度小于 u，则根据 u 来生成一个新的 turbo_shift 偏移量，取 delta1、delta2 及 turbo_shift 三者中的最大者来移动 P，从而使平均滑动幅度增大。为什么 turbo_shift 也可以作为一个指导性原则，下面是具体分析。我们约定 u 为记录因子，也就是前次匹配的后缀，v 为本次的已匹配后缀。显然，子串 uzv 为 P 的后缀。假设 a 和 b 是导致当前匹配失配的那两个字符，那么串 av 是 P 的后缀，同时也是 u 的后缀，因为 |u| > |v

l。如果 a 和 b 在目标串中的距离为 D，同时 P 中长度为 |uzv| 的后缀包含长度为 |zv| = D 的后缀，则在目标串中长度为 D 的子串不可能同时覆盖 a 和 b。而 u 不可能包含 b，因此最小的可移动量为 |u| - |v|，即 turbo - shift。具体如图 1 所示。

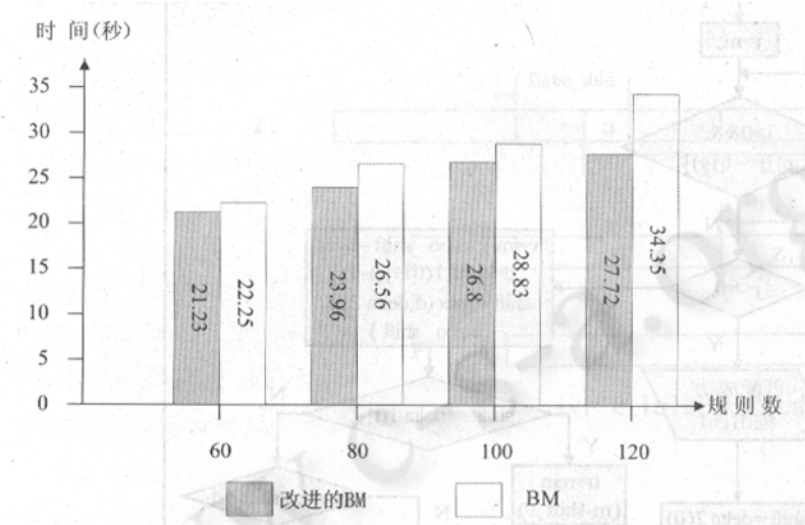

图 5　普通文本情况的时间性能比较

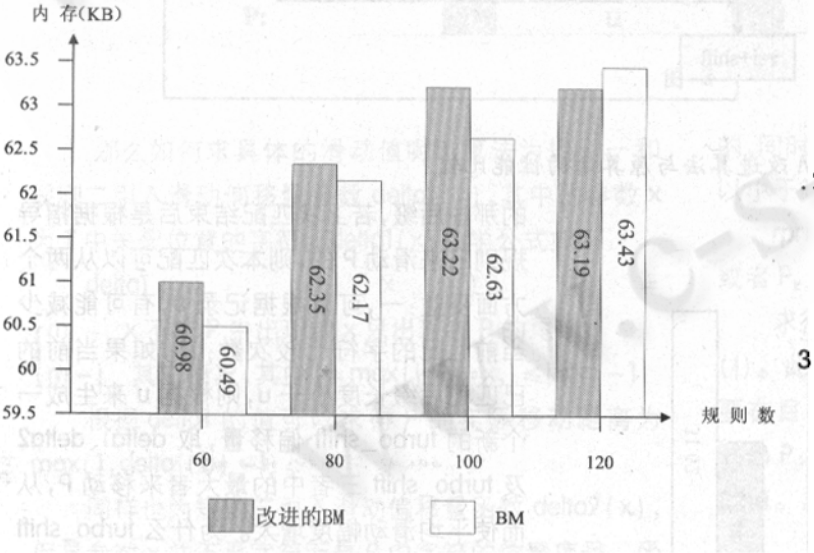

图 6　空间性能比较

同样在这种情况下，如果 delta1 的值比 delta2、turbo - shift 都要大，那么这个移动量必须大于或等于 |u| + 1。因为如图 2 中所示，c 与 d 是不相等的，因为我们的前提是上次匹配后使用的是好后缀规则。如果移动量小于 |u| + 1，则 c 与 d 将与 v 中的同一个字符对齐，那么匹配是不可能成功的。

改进的算法流程如图 3（注：数组的下标从 0 开始）。

时间性能的比较，这里先选用重复后缀较多特殊文本 60 条，然后依次增加类似文本 20 条来进行检测。从图 4 可以看性能改善的比较明显。

然后把特殊文本全部换成随即选取的普通文本，检测方法同刚才一样，从图 5 可以看出性能有一定的改善，但是不明显。

接下来两种算法内存使用量的比较，如图 6 所示。

以上的实验数据表明，虽然系统牺牲了一些额外的预处理时间和内存空间，但是在重复后缀较多的情况下能够明显的加速匹配过程，因而这些牺牲是相当值得的，即便是在一般情况下，也能带来一定的性能改善。

参考文献

1 BOYER RS, MOORE JS A fast string searching algorithm[j], Communications of ACM, 1977, 20(10): 762 - 772.

.2 KNUTH DE, MORRIS J H, PRATT VR. Fast pattern matchingin string[J], SIAMJournal on Computing, 1977, (6): 323 - 350.

3 http://www - igm. univ - mlv. fr/ ~ lecroq/string/node14. html#SECTION00140.