



一种关键字表达式的匹配优化方法^{*}

杨天龙, 张宏莉

(哈尔滨工业大学计算机科学与技术学院计算机网络与信息安全技术研究中心 哈尔滨 150001)

摘要:随着网络安全应用需求的增加,内容检测系统中特征模式集规模不断增长,模式间的冗余重复越来越多。大量模式间的冗余重复直接影响了系统的资源使用率和检测性能。本文首先对表达式的包含关系做了分类,提出了表达式冗余消除算法,然后在 BitCount 算法的基础上提出了一种时间复杂度为 $O(1)$ 的掩码验证算法 MaskVeri。实验结果表明,改进后的算法性能更高。

关键词:表达式匹配;掩码验证优化;表达式消除

doi: 10.3969/j.issn.1000-0801.2013.01.006

Optimization of Expression Matching for String Matching

Yang Tianlong, Zhang Hongli

(Computer Network and Information Security Technology Research Center,

School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

Abstract: As the requirement of network security application increases, more and more redundant patterns appear and the patterns' amount of intrusion detection system (IDS) raises. The redundant patterns affect the coefficient of utilization of system's resource, and the performance of IDS directly. The inclusion relationship for expression was classified, and the algorithms eliminating redundant expressions were proposed. Then, to optimize the BitCount algorithm, a theoretical $O(1)$ algorithm (MaskVeri) for mask verification was designed. The experimental result shows that MaskVeri performs better than other programming realization.

Key words: expression matching, mask verification optimization, expression elimination

1 引言

串匹配技术^[1-3]是入侵检测、病毒防护、信息内容监测等网络安全系统中的关键技术之一,直接影响检测特征集的可配置规模和检测的实时性。但单关键字匹配并不能准确地体现用户所关注的内容。例如用户希望统计同时存在“郭美美”和“红十字会”这两个关键字的网页,这就需要使用与表达式:“郭美美”&“红十字会”。表达式匹配需要解决两个主要问题:表达式消冗和快速匹配。

之所以需要对表达式消冗,是因为表达式各关键字之

间或各表达式之间部分存在包含关系。例如在表达式“apple&banana&ppl”中,关键字“apple”包含了“ppl”,去除这个关键字“apple”并不会影响系统的检测准确性。大量存在的表达式冗余,一方面因额外占用系统内存而导致模式集可配置规模降低,另一方面由于重复匹配而降低了系统检测性能。

内容安全检测系统中的表达式消冗优化本质上可归结为布尔表达式化简问题,参考文献[4]中定义了最简表达式的4种性质:表达式是若干个字符串的析取形式;每一个字符串中不应有重复字符并按字典序排好;对于各个字

^{*} 国家重点基础研究发展规划(“973”计划)基金资助项目(No.2011CB302605),国家自然科学基金资助项目(No.61173145)



字符串出现的顺序,字符串长度短的排在长的前边,长度相同的字符串按照字典序进行排序;如果字符串出现在表达式中,则该串的任何超集都不能出现在表达式中。本文中所讨论的针对网络环境下特征集的表达式的化简属于参考文献[4]中的“超集串”范畴的化简。

研究者在布尔表达式化简方面已开展了大量研究工作^[4-6]。其中,最具代表性的表达式化简技术是 Karnaugh map (K-map)^[7]和 Quine-McCluskey (QM)^[8]。在 K-map 算法中,参考文献[5]通过牛顿二项展开式计算 n 变量 K-map 分组的数目,以期通过减少分组总数达到降低布尔代数表达式数目的目的。在匹配性能方面,QM 算法优于 K-map 算法。参考文献[6]提出了针对 QM 的优化方法 MQM,该方法使用代数方法减少最小项的比较次数,用 E-Sum 降低最小项中的重复项。总体上,针对 K-map 和 QM 的优化均通过减少布尔表达式的数目来提高布尔表达式的性能,在逻辑比较复杂的电路化简等领域取得了显著的应用效果。

随着网络流量的不断增长,网络安全需求的不断增加,近年来多关键字表达式的匹配性能问题受到了广泛关注。2007 年中国科学院的曹京、谭建龙等人提出了布尔表达式匹配通用算法框架^[9]。2007 年中国科学院曹京、刘燕兵等人分析了关键字表达式匹配的两种主要技术:双映射表和 BitCount 技术^[10]。随后又特别针对定序布尔表达式的匹配方法设计了算法。2011 年哈尔滨工业大学张树壮设计了基于 XFA 的算法^[11],弥补了参考文献[10]中算法无法表示元素之间相对位置的缺陷。2012 年宋云、龙际珍则针对

有限窗口内的布尔表达式匹配提出了一种时空复杂度均为 $O(1)$ 的算法^[12]。

本文主要研究网络信息内容检测中与表达式匹配的冗余消除问题以及如何提高掩码验证的效率。

2 冗余表达式消除方法

2.1 用于描述表达式的定义

表 1 中列出了文中用到的符号。

在本文中将对表达式和表达式中一些相关的元素做如下定义。

定义 1 表达式定义: $BE = \{be_1, be_2, \dots, be_n\}$, 其中, BE 是布尔表达式的集合, be_i 是一个布尔表达式, $be_i = key_{i,1} \& key_{i,2} \& \dots \& key_{i,|be_i|}$, key_i 是规则集中的一个关键字, 是字符集 Σ 上的有限字符序列。 $|be_i|=1$ 时, 称表达式 be_i 为单关键字表达式, 此时直接用 $key(be_i)$ 表示该表达式的关键字; $|be_i|>1$ 时, 称表达式 be_i 为多关键字表达式。 $BE = BE_{multiple} \cup BE_{single}$, $\forall b \in BE_{multiple}, |b|>1, \forall b \in BE_{single}, |b|=1$ 。 BE_{single} 称为 BE 的单关键字集合, $BE_{multiple}$ 称为 BE 的多关键字集合。本文中不对表达式中关键字在文本中出现的位置做要求。令 $|key_{i,m}|$ 表示关键字 $key_{i,m}$ 的长度, $len(be_i) = \sum_{m=1}^{|be_i|} |key_{i,m}|$ 定义为表达式 be_i 的字符串长度。 $\min len(be_i) = \min(len(be_i))$, 即 BE 中具有最短字符串长度的表达式的长度。

定义 2 表达式冗余消除规则: $\exists SBE \subseteq BE$ 和 $be_i \in BE, \forall s \in SBE, be_i > s$ 。设 $\min len BE(SBE) = \{e | e \in SBE \text{ 且 } len(e) = \min len(SBE)\}$, $\min len BE(SBE)$ 中的任何一个表达式都可

表 1 符号定义

符号	符号含义	符号用例
BE	表达式集合	见定义 1 中
be	表达式	见定义 1 中
$key_{i,j}$	表达式中的关键字	$key_{i,j}$ 表示第 i 个表达式中第 j 个关键字
$key(be_i)$	单关键字表达式的唯一关键字	见定义 1 中
$\ $	关键字的长度	$ key_{i,j} $
	表达式中所有关键字的长度和	$ be $
$>$	表达式的包含关系	如 $be_m > be_n$ 表示表达式 be_m 包含表达式 be_n
∇	表达式的不包含关系	如 $be_m \nabla be_n$ 表示表达式 be_m 不包含表达式 be_n
$\triangleright$	字符串包含关系	如 $txt_m \triangleright txt_n$ 表示 txt_m 是 txt_n 的子串
$\&$	两个关键字在表达式中同时存在	表示 $key_m \& key_n$ 关键字 key_m 和关键字 key_n 同时存在

作为替换 be_i 的表达式。

2.2 表达式包含关系分类

为消除冗余表达式,首先需要将布尔表达式的包含关系进行归类。根据冗余现象出现于多少个表达式之中,可以将表达式包含关系分为如下2类。

第1类:表达式内部的包含关系, be_i 中, $\exists 0 \leq m, k \leq |be_i|$,且 $m \neq k$,使得 $key_{i,m} \supset key_{i,k}$,可删除 $key_{i,k}$ 。例如表达式“abcdefg”&“bcd”&“xyddwrwe”中,由于“bcd” $\supset$ “abcdefg”,整条表达式可以简化成“bcd”&“xyddwrwe”。

第2类:表达式之间的包含关系。这种情况可以具体分为如下5种。

①单一关键字表达式之间的包含关系, $\exists be_i = key_{i,1} \in BE$ 和 $\exists be_j = key_{j,1} \in BE, |be_i| = |be_j| = 1$,使得 $key_{i,1} \supset key_{j,1}$ 。此时, $be_j > be_i$,可删除 be_j ,例如“abcdefg”和“bcd”。

②单一关键字表达式包含多关键字表达式, $\exists be_i = key_{i,1} \in BE$ 和 $\exists BE_j = key_{j,1} \in BE, |be_i| = 1, |be_j| > 1$,但是 be_j 可以由 be_i 识别。此时, $be_j > be_i$,可删除 be_j 。例如“abcdefg”和“bc”&“ef”。

③多关键字表达式中的某一个关键字包含单一关键字表达式, $\exists be_i = key_{i,1} \in BE$ 和 $\exists BE_j \in BE, |be_i| = 1, |be_j| > 1$,且 $\exists 1 \leq m \leq |be_j|$,满足 $key_{i,1} \supset key_{j,m}$ 。此时, $be_j > be_i$,可删除 be_j 。例如“ab”和“abc”&“def”。

④两个多关键字表达式中,一个表达式的某个关键字能够被另一个表达式识别, $\exists be_i \in BE$ 和 $\exists be_j \in BE, |be_i| > 1, |be_j| > 1$,且 $\exists 1 \leq m \leq |be_j|$,满足 $key_{j,m}$ 可由 be_i 识别。此时, $be_j > be_i$,可删除 be_j 。例如“xyz”&“abcdefg”和“bc”&“ef”。

⑤除①~④之外其他各种复杂的表达式包含关系。可以表示成两个表达式都是多关键字表达式,一个表达式的关键字子集能够被另一个表达式代表, $\exists be_i \in BE$ 和 $\exists be_j \in BE, |be_i| > 1, |be_j| > 1$,如果 $\forall key_{i,m}, 1 \leq m \leq |be_i|, \exists key_{j,t}, 1 \leq t \leq |be_j|$,满足 $key_{i,m} \supset key_{j,t}$,且 $len(be_i) \leq len(be_j)$ 。此时 be_j 可以被 be_i 识别。此时, $be_j > be_i$,可删除 be_j 。例如“abh”&“bcd”&“efg”和“ab”&“bc”。

为了尽可能消除表达式中的冗余信息,本文仅保留存在包含关系的表达式中具有最短表达式长度的表达式。

在以上6种包含关系中,第1类和第2类中的情况①是比较常见的基本包含关系。本文仅讨论这两种情况下表达式的消冗方法,其余几种情况可以在第2类中的情况①的处理方法的基础上进行扩展解决。

2.3 针对第2类中的情况①形式的表达式的冗余消除

这种情况下可通过构造自动机(如AC算法)来消除冗余表达式,具体算法如下。

算法1 利用AC自动机实现单关键字表达式的冗余消除算法

输入:Single Key Expression Set

输出:经过精简的单关键字表达式集合

AC trie= $\emptyset$

List=Sort Single Key Expression according to length's growth

For each gjz $\in$ List

If gjz can be recognized by AC Then

mark gjz as discarded

Else

add gjz to AC

mark gjz as reserved

EndIF

EndFOR

这种方法仅适合于关键字规模较小时使用,当关键字规模达到AC内存极限时将无法完成自动机的构造。由此本文设计了一种利用散列的单关键字表达式消除冗余的方法。为了能达到最好的效果,需要对单关键字表达式做如下处理。

步骤1 令 $BE_{single} = \bigcup_{i=1}^m SBE_i, \cap SBE_i = \emptyset, \forall i, 1 \leq i \leq m$, 即针对子集 SBE_i , 存在一个固定常数 $setlen(i), \forall a, b \in SBE_i, len(a) = len(b) = setlen(i)$, 且 $\forall 1 \leq i < j \leq m$, 有 $setlen(i) < setlen(j)$ 。

步骤2 设 M 和 m 分别为所有单关键字的最大长度和最短长度。hashtable是一个长度为HashTableSize的散列表。SHIFT是一个双字符位移表,实现机制和WM中的SHIFT表一样。

依次将 i 从 m 增加到 M 并按照如下过程处理每个子集 SBE_i 。

(1) $\forall b \in SBE_i, hv = hashCompute(key(b), m)$, 将 $key(b)$ 插入散列表入口处 hashtable[hv]的链表头,原链表中的内容顺次后移。

(2) 通过 $(key(b))[0, 1, \dots, m-1]$ 中的 $(key(b))[x, x+1], 0 \leq x \leq m-2$, 填写 SHIFT。

经过步骤2后每个散列表入口处 hashtable[hv]的链表 link(hashtable[hv])中的关键字已经按照长度升序排序。



步骤 3 对 $link(hashable[hv])$ 中每个长度段的关键字按照 ASC II 码顺序排序。

经过以上处理,可利用如下算法 1 对单关键字表达式集合进行消冗。

算法 2 单关键字表达式之间的冗余消除算法

输入: hashtable, $BE_{single} = \bigcup_{i=1}^m SBE_i$

输出: 经过精简的单关键字表达式集合

For $i=M$ to m

$\forall b \in SBE_i, i=m-2, \min = 10\ 000, id=-1$

If $SHIFT[b[i]][b[i+1]]$ 为 0 Then

$link = hashtable[hashCompute(\&b[i], m)]$

$leRe = len(b) - i + m - 2$

While link

If $len(link) \geq leRe$ Then

break

EndIF

If $cmp(link, \&b[i], len(link)) == 0$ Then

delete $gjz_id(link)$ from BE_{single}

break

EndIF

$link++$

EndWHILE

$i++$

Else

$i += SHIFT[b[i]][b[i+1]]$

EndIF

EndFOR

算法复杂度: 由于算法采用了类似于 WM 算法的 SHIFT 表, 且判断过程类似于 WM 匹配算法, 因此算法在理论上具有 $O(n/m)$ 的复杂度, 其中, n 为每条被扫描的单关键字表达式的长度, m 为最短单关键字表达式的长度。此外经过步骤 1~步骤 3 的处理过程, 散列表入口处的单关键字表达式被按照长度进行了排序, 通过对关键字当前字符指针后剩余子串长度和散列表链表中关键字长度的比较(算法第 7 行), 可以排除一部分不需要比较的关键字, 尽量加快删除“含有包含关系的两个单关键字表达式”中的较长的关键字的过程。

2.4 针对第 1 类的表达式的冗余消除

这种形式的表达式 BE_i 中, 只要删除含有子串关系的两个关键字中的较长者就可以实现表达式冗余消除。

算法 3 单个多关键字表达式中的冗余消除算法

输入: $BE_i = key_i^1 \& key_i^2 \& \dots \& key_i^{itemcount(BE_i)}$

输出: 经过精简的表达式 newbds

For $m=1$ to $itemcount(BE_i)$

For $n=m+1$ to $itemcount(BE_i)$

If $key_i^m \supset key_i^n$ Then

标记 key_i^m 可被删除

EndIF

If key_i^n 是 key_i^m 的子串 Then

标记 key_i^n 可被删除

EndIF

EndFOR

EndFOR

创建不包含关键字的新表达式 newbds

For $m=1$ to $itemcount(BE_i)$

If key_i^m 未被标记可删除 Then

将 key_i^m 加入 newbds

EndIF

EndFOR

以上两个算法是消除含有形如第 1 类和第 2 类中的情况①样式的表达式的方法。这两种可消除的表达式是各类表达式中最基本的类型, 对第 2 类中的情况①的处理方法进行扩展后, 并结合关键字匹配、表达式匹配就可以完成对第 2 类中的情况②~情况⑤样式的表达式的冗余消除。本文不做详细说明。

由于这部分中的表达式冗余消除算法的性能实际上取决于采用的字符串匹配算法的性能, 因此本文中没有给出实际测试数据。

3 掩码表达式验证优化

3.1 优化算法

基于关键字表达式的匹配方法在参考文献[10]中已有详细说明。本文针对参考文献[10]中的普通 BitCount 算法中的掩码处理环节给出了一种理论性能上更加高效的实现方法, 以期进一步减小表达式处理开销。

根据文献中描述, 匹配表达式需要两个主要的数据结构: 表达式信息表和关键字信息表。为适应 BitCount 算法, 可以在表达式中保存一个比特掩码, 用来表示这个表达式中共含有多少个关键字; 关键字信息中则需要保存该关键字出现在哪些表达式中。详细处理方法可以参照参考文献[10]。

第一种表达式掩码结构 <pre>typedef struct expression{ unsigned char BEMask [BDS_ITEM_MAX]; }EXP1;</pre>	关键字出现所属表达式的链表 <pre>typedef struct gjz{ ABL*appear_bds_list; }GJZ;</pre>
第二种表达式掩码结构 <pre>typedef struct expression{ unsigned char BEMask [BDS_ITEM_MAX]; unsigned char CurMask [BDS_ITEM_MAX]; }EXP2;</pre>	用于存储匹配过程中出现的表达式的链表 <pre>typedef struct bds_address{ EXP2 * changed_mask_address; struct bds_address*next; }BDSADD;</pre>
关键字在表达式中的位置信息 <pre>typedef struct gjz_offset{ int bds_sn; int offset_in_bds_sn; struct gjz_offset*next; }ABL;</pre>	

图 1 BitCount 算法中主要的数据结构

BitCount 算法中主要的数据结构如图 1 所示。

利用 BitCount 匹配表达式时, 可以有两种处理方式: 表达式数据结构中不存放当前布尔表达式的计算结果 CurMask, 只在实际匹配时才计算 CurMask, 即 EXP1 数据结构; 表达式数据结构中存放 CurMask, 即 EXP2 数据结构。为了比较两种方法, 本文给出了 BitCount 表达式匹配的两种代码。

算法 4 链表实现的掩码验证算法

输入: 文本 text, 表达式集合 EXP1 bds_array, 关键字集合 GJZ gjz_array;

输出: 匹配到的表达式

申请临时文本指针 sPtr=text

临时表达式掩码指针集合 EXP1 *temp_set=NULL

While endoftext(text)-sPtr

 gjz_id=StringMatch(sPtr)

If gjz_id 合法 Then

 For node=gjz_array[gjz_id]

 在 temp_set 中寻找 node.bds_sn 的结果→ptr

 If ptr==NULL Then

 将 node.bds_sn 加入 temp_set 中, 并将这个指针赋

给 ptr

 EndIF

 ptr.CurMask|=node.offset_in_bds_sn

 If ptr.CurMask == ptr.BEMask Then

 报告匹配表达式 node.bds_sn

 EndIF

EndFOR

EndIF

EndWHILE

Release all element in temp_set

算法 5 MaskVeri 掩码验证算法

输入: 文本 text, 表达式集合 EXP2 bds_array, 关键字

集合 GJZ gjz_array;

输出: 匹配到的表达式

申请临时文本指针 sPtr=text

申请表达式掩码指针集合 BDSADD *pointer_set =

NULL

While endoftext(text)-sPtr

 gjz_id=StringMatch(sPtr)

If gjz_id 合法 Then

 For node=gjz_array[gjz_id]

 If bds_array [node.bds_sn].CurMask==0 Then

 BDSADD *newadd.changed_mask_address =

 &(bds_array [node.bds_sn])

 将 newadd 加入 pointer_set

 EndIF

 bds_array [node.bds_sn].CurMask|=

 node.offset_in_bds_sn



```

If bds_array [node.bds_sn].CurMask ==
bds_array [node.bds_sn].BEMask Then
    报告匹配表达式 node.bds_sn
EndIF
EndFOR
EndIF
EndWHILE
While pointer_set
    pointer_set.CurMask=0
    pointer_set=pointer_set.next
EndWHILE
Release all element in pointer_set

```

3.2 复杂度分析

为了叙述方便,本文用 $gjzBdsSet(gjz_id)$ 表示一个表达式集合,该集合中的每一个表达式都包含关键字 gjz_id 。

通过比较两种算法可知:采用 EXP1 数据结构的算法在匹配到关键字 gjz_id 后,需要检查该关键字所属表达式的信息是否已经存入用于临时存放表达式信息的 $temp_set$ 指针集合中,这里会有查找表达式的开销。令 $MatchedGjzSet$ 为搜索某一文本时找到的关键字集合, $\forall id \in MatchedGjzSet$, 那么匹配到关键字后要搜索的集合中的表达式个数为 $e = |U gjzBdsSet(id)|$ 。排除关键字匹配开销、掩码设置及掩码测试的开销以及最后临时数据的释放开销,采用 EXP1 数据结构的表达式匹配过程中的处理时间为 $O(search(U gjzBdsSet(id)))$, 具体开销和采用的搜索方法有关。

采用 EXP2 的算法中加入了一个额外的数据结构 BDSADD, 用来保留已被成功匹配的关键字所涉及的表达式的信息 ($U gjzBdsSet(id)$) 的数据结构。当匹配成功一个关键字后,对于该关键字出现的所有表达式,仅通过算法第 7 行中判断表达式的 $CurMask$ 是否为 0, 如果非 0, 表示该表达式已被存储于 BDSADD 数据结构中; 如果为 0, 表示该表达式还未被处理过。比较算法 4, MaskVeri 算法在处理成功匹配的关键字时无需进行集合搜索, 其处理时间为 $O(1)$ 。MaskVeri 算法在处理过文本后只需通过 BDSADD 数据结构中的信息将被改动过的表达式中的 $CurMask$ 重新设置为 0, 并释放临时数据信息。

4 实验结果

本文的实验运行于一台配备了 Pentium Intel Core 2 Duo 双核处理器, 主频 2.0 GHz 的笔记本上, 物理内存 2 GB,

一、二级缓存分别为 32×2 KB 和 2 048 KB。实验主机操作系统为 Fedora 16 x86-64 Linux, 其内核为 3.1.0。编译环境为 gcc 编译器 4.6.2, 编译时开启了 -O3 选项。

随机构造布尔表达式 10 000 个, 每个表达式包含 2~4 个关键字。关键字随机生成, 关键字长度范围为 16~21 byte。生成关键字的字符来自于 ASCII 码中的可见字符集。测试文本数据长度为 10 MB, 先随机选择需要撒入文本数据中的表达式 (按照撒入比例和被随机选择到的表达式中关键字的字符总数确定具体选择的表达式数目, 撒入比例为 10%~100%, 表示所选表达式的所有关键字的字符总数和产生的总文本的字符数的比值, 步长为 10%), 然后生成除去撒入表达式总长度后的随机数据。最后将这些表达式包含的关键字平均并且随机地插入上述生成的随机数据中, 这样就生成了测试文本数据。本文采用了一种基于 SSE 的匹配方法进行底层关键字匹配。测试中, 本文除了使用了算法 3 和算法 4 中描述的处理方法外, 还额外使用二元查找树的方法实现了一个掩码测试的方法, 由于篇幅所限, 本文没有将这种方法以算法的形式在文中说明。具体的测试结果如图 2 所示, 随着撒入比例的增加, 各种掩码判断方法性能都有所下降。本文设计的 MaskVeri 掩码验证方法最优, 其性能优于二元查找树和链表式验证方法的实测效果。

5 结束语

为了降低表达式匹配占用的系统资源, 本文分析了表达式的各种包含关系, 并针对其中最基本的包含关系提出了 2 种表达式冗余消除算法, 这 2 个算法的复杂度不高于串匹配算法。然后为了提高表达式匹配性能, 在 BitCount 算法的基础上提出了一种时间复杂度为 $O(1)$ 的掩码验证算法 MaskVeri。实验结果表明 MaskVeri 算法性能更优于链表式验证方法和二元查找树验证方法。

参考文献

- 1 Aho A, Corasick M. Efficient string matching: an aid to bibliographic search. Communications of the ACM, 1975; 333~340
- 2 Snort 2.4.x. <http://www.snort.org>, 2006
- 3 Clam antiVirus. <http://www.clamav.org>, 2006
- 4 王德才, 徐建国, 吴哲辉等. 布尔表达式的化简与并行排序网络验证. 计算机工程与设计, 2009(14): 3322~3325
- 5 Asli B H S. Survey of a new note on karnaugh maps. International Journal of Control and Automation, 2012, 5(2): 21~24
- 6 Jadhav V, Buchade A. Modified quine-McCluskey method. CoRR,

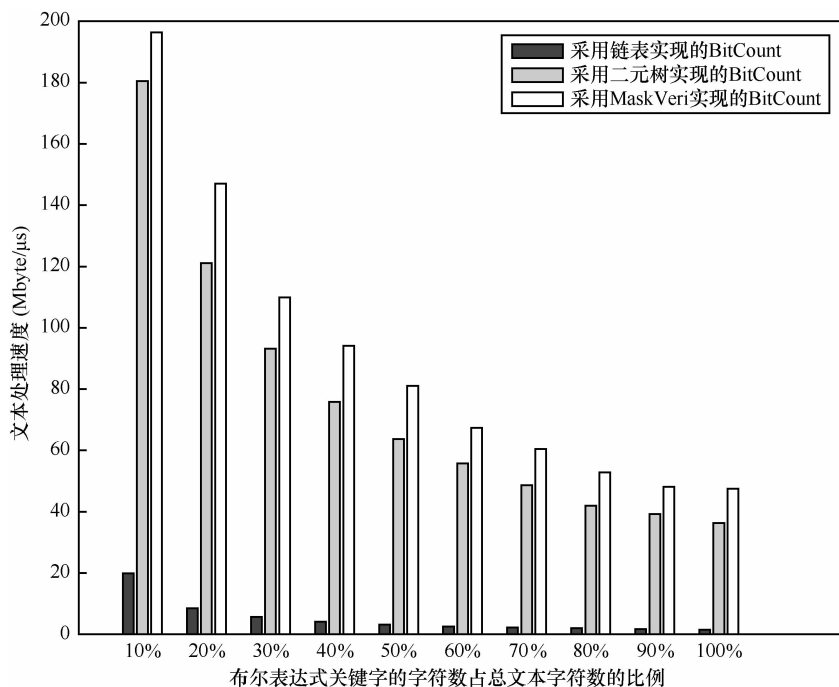


图2 3种掩码表达式验证方法性能比较

2012(2289): 1~4

- 7 Karnough M. The map method for synthesis of combinational logic circuits. AIEE, 1953:593~599
- 8 McCluskey E J. Minimization of Boolean function. Bell System Tech Journal, 1956, 35(5):1417~1444
- 9 曹京,谭建龙,刘萍等.布尔表达式匹配问题研究.计算机应用研究,2007(9): 70~72,108
- 10 曹京,刘燕兵,刘萍等.定序窗口布尔表达式匹配技术研究.通信学报,2007(12):125~130

11 张树壮.面向网络安全的高性能特征匹配技术研究.哈尔滨工业大学博士学位论文,2011

12 宋云,龙际珍.规则数量无关的多布尔表达式匹配算法.软件导刊,2012(3):57~59

【作者简介】 杨天龙,男,哈尔滨工业大学博士研究生,主要研究方向为大规模特征集串匹配技术。张宏莉,女,博士,哈尔滨工业大学教授、博士生导师,主要研究方向为网络与信息安全、网络测量与建模、网络计算、并行处理等。

(收稿日期:2012-09-19)

(上接第38页)

的应用,不仅为LTE网络优化提供了分析工具,也为网络监测和质量评估提供强有力的支撑。

参考文献

- 1 唐海.TD-LTE的标准化工作及进展.电信科学,2009(1): 22~26
- 2 3GPP TS 36.300.Evolved Universal Terrestrial Radio Access Network (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN),Overall Description V8.12.0, 2010
- 3 3GPP TS 36.410.Evolved Universal Terrestrial Access Network (E-UTRAN), S1 General Aspects and Principles V8.3.0, 2010
- 4 魏辉,张治中.TD-SCDMA网络测试仪中SCCP解码及上层PDU获取方案.重庆邮电大学学报(自然科学版),2007,19(1):

47~52

- 5 3GPP TS 36.413. Evolved Universal Terrestrial Radio Access Network (E-UTRAN):S1 Application Protocol V8.10.0, 2010
- 6 夏韪,雒江涛,张治中.TD-SCDMA测试仪中Iub接口CDR的合成方案.重庆邮电大学学报(自然科学版),2007(2):35~38
- 7 陈玉花,张治中,左书川等.TD-SCDMA网络Iu-PS口CDR合成方案研究.电信科学,2009(11):50~54
- 8 舒忠玲.基于文本配置文件的多段关联方法.中国:CN 101527653A, 2009

【作者简介】 李艳,女,重庆邮电大学硕士研究生,主要研究方向为通信协议及软件、LTE网络测试技术。张治中,男,重庆邮电大学教授、博士生导师,主要研究方向为第三代移动通信测试技术、宽带信息网、NGN、LTE网络等。

(收稿日期:2012-10-28)