ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.1, January 2007, pp.117−126
DOI: 10.1360/jos180117

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于有状态 Bloom filter 引擎的高速分组检测*

叶明江+, 崔 勇, 徐 恪, 吴建平

(清华大学 计算机科学与技术系,北京 100084)

Fast Packet Inspection Using State-Based Bloom Filter Engine

YE Ming-Jiang+, CUI Yong, XU Ke, WU Jian-Ping

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

+ Corresponding author: Phn: +86-10-62785822, Fax: +86-10-62788109, E-mail: yemingjiang@csnet1.cs.tsinghua.edu.cn

Ye MJ, Cui Y, Xu K, Wu JP. Fast packet inspection using state-based Bloom filter engine. *Journal of Software*, 2007,18(1):117−126. http://www.jos.org.cn/1000-9825/18/117.htm

Abstract: More and more network security applications depend on inspecting the content of the packets to detect the malicious attacks. To detect these attacks online, packet inspection demands exceptionally high performance. A lot of research works have been done in this field, and yet there is still significant room for improvement in throughput and scalability. This paper proposes a fast packet inspection algorithm based on state-based Bloom filter engines (SABFE). To achieve high throughput, parallel design is adopted when searching in one Bloom filter engine and between multiple Bloom filter engines. In addition, specific lookup table and prefix register heap are constructed in SABFE to keep the state of the matched prefix for the sake of detecting long patterns. The analysis and the evaluation show that the high throughput of the algorithm can satisfy the wire speed detection requirement when the low resource consumption in hardware resource further improves the scalability of SABFE.

Key words: network security; NIDS (network intrusion detection systems); packet inspection; pattern matching; Bloom filter

摘 要: 越来越多的网络安全技术通过分析网络分组中的内容来检测报文中是否含有恶意攻击代码.为了能够在线检测攻击,部署在路由器中的分组检测模块对于分组检测的速度也提出了越来越高的要求.虽然在这个领域已有很多研究工作,然而在性能、可扩展性和适用性方面还有很多可研究的空间.提出了一种基于有状态 Bloom filter 引擎的高速分组检测方法 State-Based Bloom filter engine(SABFE).通过并行查找 Bloom filter 和前缀寄存器堆,以及利用多个并行的 Bloom filter 引擎进行流并行检测,达到了较高的吞吐性能.同时,利用快速查找表和前缀寄存器堆保存当前子串的匹配状态来检测长的规则.分析和模拟实验表明:该方法在规则长度增加时依然保持了较高的吞吐性能,可以实现线速的分组检测,同时,极大地减少了硬件资源开销,提高了可扩展性.

关键词: 网络安全;网络入侵检测;分组检测;串匹配;Bloom filter
中图法分类号: TP393 文献标识码: A

Internet 开放的结构以及设计之初并没有对安全作很多考虑,因而对网络上恶意的攻击,包括病毒、蠕虫[1,2]

* Supported by the National Natural Science Foundation of China under Grant Nos.60473082, 60403035 (国家自然科学基金); the National Grand Fundamental Research 973 Program of China under Grant No.2003CB314801 (国家重点基础研究发展规划(973))
Received 2005-11-08; Accepted 2006-04-03

等缺乏必要的约束.最近,研究者开始通过分析报文的内容,而不仅仅是报文的头部来检测病毒和蠕虫等恶意攻击[3,4]. 他们往往利用在端系统、接入路由器或者核心路由器部署新的功能来在线检测过滤恶意代码[5],保护网络和端系统.当前,高速分组检测系统面临的最大挑战是网络传输速率的日益增加和攻击方式的多样性.评价分组检测的算法往往需要考虑以下几方面:(1) 速度.在线检测必须达到线速,避免检测系统本身成为网络攻击的目标;(2) 可扩展性.包括规则集大小和规则长度的可扩展性.攻击方式的增加导致规则集日益增大,同时,复杂蠕虫和病毒的特征码也往往达到上百字节.所以,检测系统应该能够支持大规模的规则集合和长的特征字符串;(3) 开销.为了实际应用,方案使用的资源必须是可以合理的,硬件实现的方案往往受到存储器容量和逻辑复杂度的限制.

基于 Bloom filter[6]的深度流检测方法[7,8]资源使用较低,同时能够满足检测系统对处理速度和支持较大规则集的要求.然而,在处理较长的特征字符串时存在可扩展性问题.本文提出了基于有状态 Bloom filter 引擎高速分组检测算法 SABFE(state-based Bloom filter engine),将状态机的思想引入到使用 Bloom filter 进行分组检测中,构造了快速查找表以保存状态机信息,同时,利用前缀寄存器堆保存当前匹配的中间状态信息,从而解决了可扩展性的问题.另外,通过并行查找 Bloom filter 和前缀寄存器堆,以及利用多个并行的 Bloom filter 引擎进行流并行检测,达到了较高的吞吐性能,满足了线速处理的需求,同时减少了硬件资源开销,具有一定的可扩展性.

1 Bloom filter

Bloom filter[6]的基本原理是:对大小为 n 的规则集中每一个字符串 X,利用 k 个 hash 函数,计算出 k 个 hash 值,值域为[0,m−1];对每一个值,将对应的 m 个 bit 长度数组的对应位置设为 1,如果已经是 1,就不再处理.

查询时,对字符串也用相同的 k 个 hash 函数 hash 出 k 个值,检查对应的 m 位的相应 k 位是否全为 1,若有一位为 0,则该字符串肯定不属于规则集;若全为 0,则以一定的误判率 f 判定该字符串属于规则集.

如图 1 所示,$x1$,$x2$ 为规则集中的字符串,而散列函数的个数为 3 个.$x1$ 和 $x2$ 利用散列函数生成的各自的签名,每个签名由 3 个散列值组成,然后将散列值作为索引在数组对应的位置设为 1.$y1$,$y2$ 为查询的字符串.查询时,利用相同的散列函数得到 $y1$ 和 $y2$ 各自的签名,每个签名由 3 个散列值组成;然后,通过查看数组中对应的位置是否为 1 来判断是否命中.如图 1 所示,可判定 $y1$ 肯定不属于规则集,而 $y2$ 由一定误判率判定属于规则集.

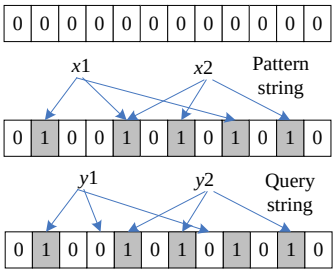

Fig.1 Bloom filter principle

图 1 Bloom filter 原理

文献[6]中,通过数学分析得到了误判率的公式:

$$f=(1-e^{-nk/m})^k \tag{1}$$

求导计算可知 f 最小满足以下条件: $f_{\min}=\left(\frac{1}{2}\right)^k$,当 $k=\left(\frac{m}{n}\right)\ln 2k$ 时.

2 基于有状态 Bloom filter 引擎的高速分组检测

2.1 系统组成

系统组成如图 2 所示,到达的分组先通过由 Bloom filter 引擎[7,8]组成的前台检测系统,该引擎在分组的内容

中对特征子串做模糊匹配.若匹配,再利用快速查找表和前缀寄存器堆构成后台分析系统进行精确分析,这是因为:(1) Bloom filter 搜索的结果并非是精确匹配的,存在一定的误检概率,虽然这个概率可以通过系统的设计降至很低[6],但无法完全消除,所以需要分析器做精确匹配;(2) 我们匹配的只是规则子串,一个长的特征串可能被分割为多个特征子串,所以需要在前缀寄存器堆中记录当前匹配的状态,从而实现长特征字符串的匹配功能.

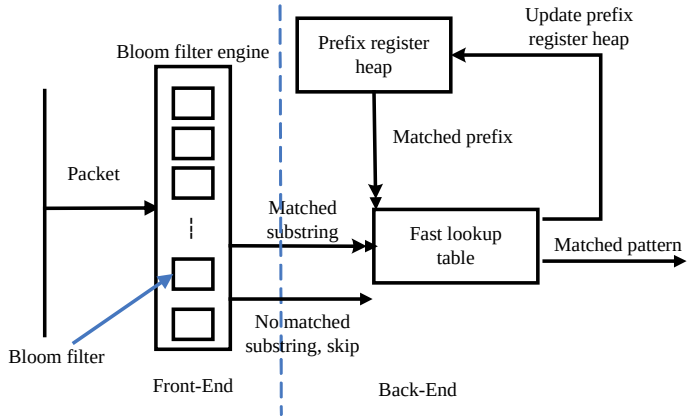

Fig.2 System overview

图 2 系统组成

在系统中,当 Bloom filter 引擎命中一个特征字符子串后,命中的子串和前缀寄存器堆中取出的当前活动前缀编号,会作为索引用来查询后台的分析器,后台分析器把 Bloom filter 引擎过滤出来的嫌疑子串做精确匹配,排除误检的字符串,然后查找命中的数据项,判断是否命中某条规则以及是否构成新的前缀,若是,则输出命中结果,并且更新前缀寄存器堆.

图 3[7]为详细的 Bloom filter 过滤引擎原理示意图.Bloom filter 引擎是在文献[7,8]中被提了出来.该引擎由一组针对特征字串长度由 L_{min} 到 L_{max} 的 Bloom filter 组成,在图 3 中,L_{min} 为 3 字节,L_{max} 为 W 字节.被检字符串每次移动一个字节,所有 Bloom filter 并行工作.如果同时有几个 Bloom filter 匹配,则按照最长字符串优先(LSF)的原则,首先把 L 值最大的字符串送分析器检查,然后,依次检查所有命中的字符串.Bloom filter 在字符串过滤应用中,其搜索时间基本上与规则集合的大小没有关系,所以能达到很高的性能.然而,因为每个特征字符串的长度都需要一个 Bloom filter 来处理,Bloom filtered 的个数随着最大特征串长度的增长而增长.

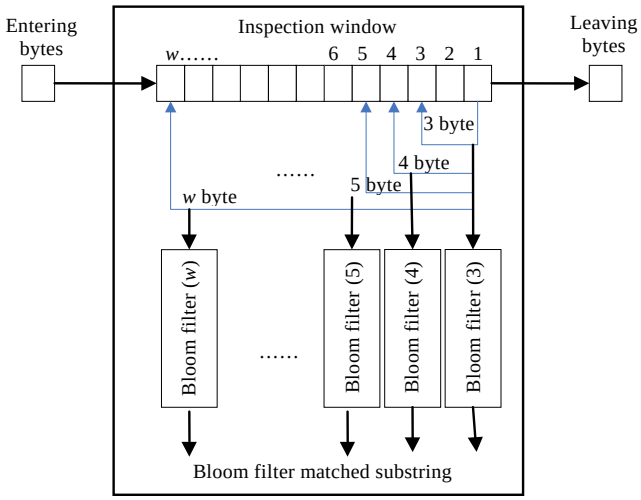

Fig.3 Bloom filter engine

图 3 Bloom filter 引擎

分析器精确匹配的性质要求它必须存储全部规则集,所以必须使用片外 SRAM 存储规则集.分析器查找采用哈希查找算法,利用链表解决哈希冲突.分析器的性能对于最后的性能有较大的影响,我们参考文献[9]中实现的基于硬件的哈希查找算法来构建后台系统,该算法能够使得哈希查找的冲突极少发生,从而获得很高的性能.

2.2 快速查找表

由于 Bloom filter 引擎能够匹配的特征串长度有限,所以,我们需要把规则分解成为子串,并且设计的数据结构保存当前可能的匹配子串和前缀串,以解决特征字符串长度的可扩展问题.保存中间匹配状态的思想在文献[10]中也使用过,然而,本文的方法与其有显著的不同:文献[10]使用 TCAM,并且使用在内存中的表保存状态,从而带来了查表的开销;本方法基于 Bloom filter,避免了 TCAM 的高功耗和高成本的问题,同时设计了以先进先出方式工作的前缀寄存器堆,保存当前活动的已匹配前缀,避免了查表的开销,同时使得数据结构的读取过程可以和分析器的精确查找过程同时进行,从而在扩展了搜索能力的同时保持很高的性能.

为了便于说明表的数据结构,假设 Bloom filter 引擎最多只能匹配长度为 L_{max}=4 的特征字符串,并以表 1 中的规则集为例进行说明.

对于表 1 中的 4 个规则,由于 Bloom filter 引擎最多只能匹配长度为 4 的特征字符串,所以,规则 2~规则 4 将无法直接检测.因此,我们需要把规则 2~规则 4 划分成不超过长度为 4 的字串,最后得到表 2 中特征字符串的子串集合.注意:拆分时长度不是 4 的倍数的规则 TFGEC 被拆成 TFGE 和 FGEC,这是因为如果我们将 TFGEC 拆分为 TFGE 和 C 的话,由于单个字符 C 在流中出现的概率非常大,会使得性能严重下降.

为了保存状态信息,经过划分之后的所有可能的前缀都被我们保存下来,见表 4.注意:前缀的长度必然是 L_{max} 的倍数,也就是 4 的倍数.此外,为了减少需要保存的信息,我们将所有的规则和所有前缀编号,在数据结构中保存编号而不是字符串.见表 3 和表 4,所有的规则和前缀串都被编号.

Table 1 Rule set
表 1 规则集合

Rule	Pattern
*R*1	ABC
*R*2	TFGEC
*R*3	TFGEMNFGET
*R*4	ABCDMNFGS

Table 2 Subtring set
表 2 子串集合

Rule	SubString
*R*1	ABC
*R*2/*R*3	TFGE
*R*2	FGEC
*R*3/*R*4	MNFG
*R*3	FGET
*R*4	ABCD
*R*4	NFGS

Table 3 Index of rule set
表 3 规则集合编号

Pattern	Index
ABC	1
TFGEC	2
TFGEMNFGET	3
ABCDMNFGS	4

Table 4 Index of prefix string
表 4 前缀编号

Prefix	Index
TFGE	1
TFGEMNFG	2
ABCD	3
ABCDMNFG	4

Bloom filter 引擎命中之后,分析器需要使用内存中的表查找,从而进行精确匹配.我们定义了内存中的扩展数据结构,用来保存当前匹配的子串的关联信息.当前 Bloom filter 引擎命中的子串,可能与已经命中的前缀组成新的前缀,也可以组成匹配的字符串.根据表 1~表 4 构造在外存中的快速查找表,见表 5.

Table 5 Fast lookup table*
表 5 快速查找表

Matched substring	Matched prefix	Index of new prefix	Index of mathched pattern	Distance between new prefix and possible suffix
ABC	*	0	1	0
TFGE	*	1	0	1,4
FGEC	1	0	2	0
MNFG	1	2	0	2
MNFG	3	4	0	1
FGET	2	0	3	0
ABCD	*	3	0	4
NFGS	4	0	4	0

已经命中的前缀编号保存在前缀寄存器堆中,它会与当前命中的字符串作为索引来对快速查找表进行查

*注:0 表示不存在或者无意义,*表示通配符.

找,从而知道当前组成的新前缀或者已经匹配的字符串的编号.同时,我们还会取出新的前缀和可能的后缀的距离数组,用来更新前缀寄存器堆.该距离数组最多有 $L_{\max}$ 个元素.例如:因为存在规则 TFGEC 和规则 TFGEMNFGET,如果命中了 TFGE 这个字串,它构成新的前缀 TFGE,编号为 1.如果移动 1 个字节就可能继续匹配后缀 FGEC,从而命中匹配字符串 TFGEC;然而,也可能在移动 4 个字节之后,继续匹配后缀 MNFG,从而形成编号为 2 的新前缀 TFGEMNFGE.

2.3 前缀寄存器堆

前缀寄存器堆设计用来保存当前的匹配状态信息.它是一个长度为 $L_{\max}$ 的寄存器堆,保存当前命中的所有有效前缀.在查询分析器的时候,我们会使用当前命中的字符子串和当前活动前缀作为索引来查找匹配表.

图 4 中,堆顶元素为编号为 1 的前缀,我们称该堆顶前缀为当前活动前缀.当前活动前缀编号是前缀寄存器堆的下一次输出,该前缀编号会和 Bloom filter 引擎命中的字符子串作为索引来寻找匹配表.

当读取前缀寄存器堆时,该堆以先进先出(FIFO)的方式工作.如图 4 所示,当堆顶的当前活动前缀 1 被读取之后,会从堆顶移走,下一个有效前缀 0 会移动到堆顶,成为当前活动前缀.

当更新前缀寄存器堆时,新的前缀编号写入寄存器堆中的对应位置即可.

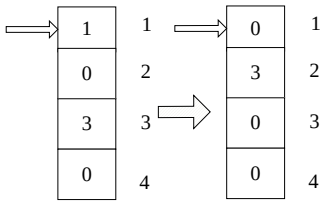

Fig.4 Prefix register heap

图 4 前缀寄存器堆

2.4 查找过程

当分组到达我们的检测器时,将字节流送入 SABFE,假设即将到来的报文内容为 TFGEC,查找过程如图 5 所示.

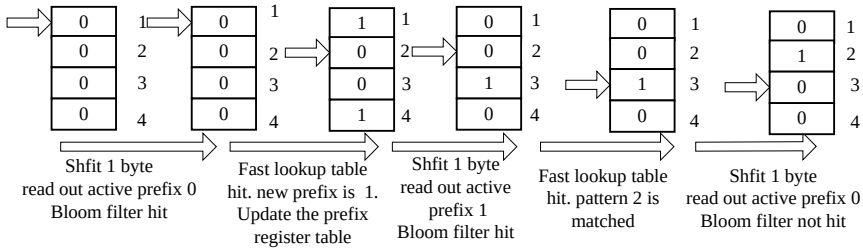

Fig.5 Detail search process

图 5 详细查找过程

(1) 开始状态前缀寄存器堆全为 0,表示没有命中的前缀;

(2) 首先,查找窗口移动一个字节.同一个时钟周期,前缀寄存器堆输出当前活动前缀编号 0;下一个时钟周期,Bloom filter 引擎命中了一个子串(TFGE);

(3) 后端系统分析器以命中的子串和当前活动前缀 0 在快速查找表中查找,命中表项 2.得知当前构成新前缀编号为 1,同时没有匹配任何规则,而可能后缀的距离是 1 和 4.于是,就会在前缀寄存器堆所对应的第 1 个和第 4 个寄存器中写入新前缀编号 1;

(4) 然后,查找窗口继续移动一个字节.前缀寄存器堆输出当前活动前缀编号 1.过滤引擎查找命中了一个子串(FGEC);

(5) 后端系统分析器以命中的子串和当前活动前缀 1 在快速查找表中查找,命中表项 3,得知当前发现 FGEC 和前缀编号 1 匹配了规则 2,也就是匹配了规则 TFGEC,但是没有构成新前缀.所以,不需要更新前缀寄存器堆;

(6) 然后,查找窗口继续移动一个字节.前缀寄存器堆输出当前活动前缀编号 0,下一个时钟周期,Bloom filter 引擎没有命中,所以不需要查找快速查找表.

2.5 流并行

为了满足对于分组检测更高速的要求,SABFE 还可以用多个器件并行搜索得到更高的性能.因为使用未经扩展的 Bloom filter 引擎进行搜索是无状态的,文献[7]中采用了字节并行的方法提供性能,也就是使用并行的器件对于同一个分组进行搜索,如图 6 所示[7].而 SABFE 由于需要保存上次的搜索结果,每次的搜索又是基于状态的,所以,SABFE 不能够对于同一个流使用多个器件并行搜索.

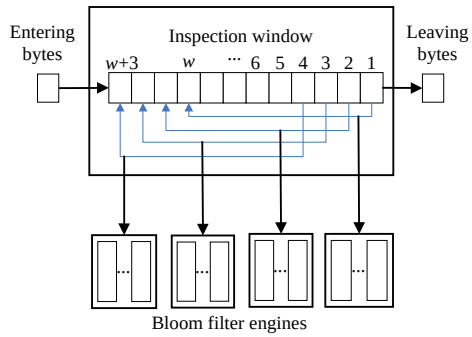

Fig.6 Byte parallel

图 6 字节并行

然而,SABFE 可以对于不同的流使用多个器件并行搜索.骨干网络中同时存在的流的数量很大,关于流分类的研究也很多[11,12].在基于流分类的基础上,对于不同的流,SABFE 可以使用多个 Bloom filter 引擎进行并行的检测.所以,采用流并行的办法仍然可以达到与字节并行同样的并行度,不会出现资源的空闲.

字节并行的好处在于:通过字节并行,不但达到吞吐量的增加,同时减少了每个分组的延时.对于 g 个并行器件而言,每个分组的延时是流并行的 $1/g$.

流并行的好处是逻辑实现简单,可扩展性好.在字节并行中,存在并行器件冲突的问题,例如:图 5 中有 3 个引擎,引擎 1 发现了感兴趣的字符串,那么需要分析器的分析是否匹配.这个时候,即使引擎 2 和引擎 3 都没有命中,如果仍然需要等待分析器完成对于引擎 1 的分析才可以移动,这样会大大降低系统的吞吐量;而流并行则不存在这个问题,每个引擎都是独立在流上工作的.

3 性能分析和比较

本节我们对 SABFE 的性能进行了分析,并进行了模拟实验.通过与其他方法进行比较,说明 SABFE 拥有高速的分组检测性能,足以满足线速的要求.同时,我们对硬件开销进行了分析和实验,说明 SABFE 在保持高吞吐检测能力的同时,解决了 Bloom filter 引擎在特征字符串长度上的可扩展性问题.

3.1 吞吐量分析

SABFE 是设计用 FPGA 或者专用 ASIC 实现的基于硬件的高速分组检测算法.耗时主要由前台的 Bloom filter 引擎的预匹配时间和后台分析系统的精确匹配时间组成.前台 Bloom filter 引擎的性能由硬件并行器件个数和硬件工作的时钟频率来决定.而后台的分析系统受到需要精确检测的字串在流中所占的比例以及内存位宽和访存速度等因素决定.

根据文献[7]及补充文献[**]中的推导,我们能够得到基于 Bloom filter 的方法在流并行时的吞吐是 $R_g=\frac{8g}{T_N+1/F}$,g 是并行的引擎个数,F 是 Bloom filter 的工作频率,N 是 Bloom filter 引擎中 Bloom filter 的个数,也就是 $N=L_{\max}-L_{\min}$, T_N 由递推式(2)得到

$$T_i=pt+(1-p)(ft+T_{i-1}) \tag{2}$$

p 是流中存在感兴趣的字符串的概率,t 是简单分析器的查找时间.

对于 SABFE 来说,所有长度小于 N 的规则的 p,t 与补充文献(见本页脚注)一样.所以, $T'_{N-1}=T_{N-1}$.而对于所有长度大于等于 N 的规则来说,都会拆分长度为 N 的特征子串,所以求解 T_N 的公式应该是

$$T'_N=p't'+(1-p')(ft'+T_{N-1}) \tag{3}$$

p'是 SABFE 中长度为 N 的子串的出现概率:

$$p'=\sum_{r\in S}\left\lceil\frac{l_r}{N}\right\rceil\left(p_r+\frac{1}{|\varepsilon|^N}\right) \tag{4}$$

证明见定理 1(见附录).r 表示规则集中的规则,l_r 表示规则 r 的特征字符串长度,$|\varepsilon|$表示字符集合的大小,p_r 表示规则 r 在流中出现的概率.

t'是后端系统分析需要的时间:

$$t'=\left(d_c+d_u+\left\lceil\frac{8N+2\lceil\log\alpha M\rceil+\lceil\log M\rceil+k\lceil\log N\rceil}{W}\right\rceil\right)\times\frac{1}{F} \tag{5}$$

证明见定理 2(见附录).W 是内存位宽,M 是规则集中规则的个数,α是划分后的规则数目和原始规则数目的比值,k 是表 5 中新前缀和可能的后缀之间的距离该项的最大数据个数,d_c 是硬件控制逻辑的时延周期,d_u 更新寄存器堆的时钟周期,F 是硬件的时钟频率.

所以,SABFE 的吞吐量为

$$R'_g=\frac{8g}{p't'+(1-p)(f't'+T_{N-1})+1/F} \tag{6}$$

3.2 性能比较

我们使用如下的环境模拟分析.原始的规则随机生成,规则的条目见表 2.匹配的规则长度从 5~L 均匀分布,L 是最长规则的长度,也是实验中观测的变化参数,用来观测在规则长度增加时性能和硬件开销的编号.其余的模拟参数见表 6.模拟的流量按照以下两条约束:(1) 每条规则在流中出现的概率相同;(2) 所有规则出现的概率之和为 p_{total},p_{total} 是实验中观测的变化参数,见表 6.

Table 6 Main simulation parameters

表 **6** 模拟中主要使用的参数

Parameters	Description	Value
g	Parallel number of Bloom filter engine	4
F	Clock frequency of FPGA	333MHZ
W	Memory word width	32bit
$L_{\min}$	Minimum length of Bloom filter engine	4
$L_{\max}$	Maximum length of Bloom filter engine	32
d_c	Hardware delay clock of control logic	3
d_u	Update clock of prefix register table	1
M	Rule number in pattern set	10000
K	Maximum number of distance between new prefix and possible suffix	8
p_{total}	The total probability of the rule pattern appear in packets	0.1 in experiment1 0.01 in experiment2

3.2.1 吞吐性能比较

模拟实验的结果如图 7 所示.从图 7 中可以看到:SABFE 虽然比基本的 Bloom filter 引擎方法中基于 Bloom

**原文献推导基于一些不准确的假设,修正后更精确的推导在 http://www.arl.wustl.edu/~sarang/analysis.pdf

filter 的方法性能略低,然而仍然保持了较高的吞吐性能.字符串检测算法有纯软件算法和基于硬件的算法.纯软件的检测方法如 KMP 算法、BM 算法、AC 算法和 Wu-Manber 算法[13]等.基于软件的算法其吞吐往往都无法达到 1Gbps 的速率,无法满足线速要求.而基于硬件的算法性能大多能满足线速的要求,如文献[14]中基于 KMP 的算法和文献[10]中基于 TCAM 的算法都能达到 1Gbps 以上的检测速度;而文献[15]中基于网络处理器实现的 AC 算法可以满足 2.5Gbps 的检测速度.从图 7 中可以看到:SABFE 能够达到 1Gbps 以上的检测速度,在网络流量中恶意流量比例为 1%时,能够达到 5Gbps 以上的检测速率.此外,SABFE 易于并行,且硬件资源开销较小,可以通过增加并行硬件资源进一步提高性能.

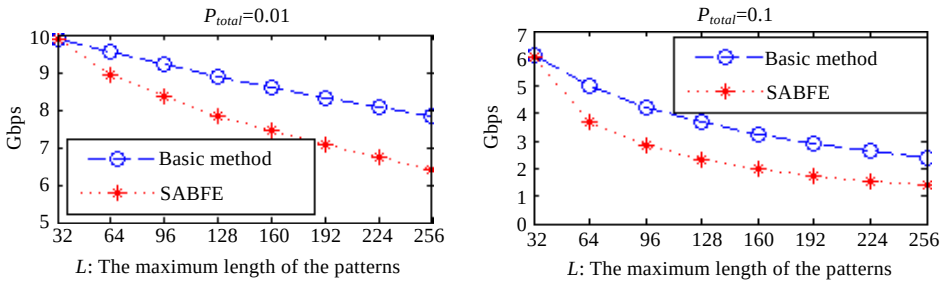

Fig.7 Comparison of performance

图 7 性能比较

SABFE 与基本的 Bloom filter 引擎方法相比会降低一些吞吐性能,因为:(1) 将长的特征串拆分为几个短的特征串,实际上,Bloom filter 需要匹配的规则数目变多,所以,流中特征字符字串的匹配概率也变大了;(2) 我们的查找表保存了状态信息,所以,分析器读取外存的周期相应增加.

因为我们的方案利用了前缀寄存器堆,使得状态记录和分析器的处理过程并行,从而保持了高的吞吐性能,满足了线速的需求.在实际系统中,大部分规则的长度都是比较短的,这种分布会使得 SABFE 平均性能更好,因为长的规则需要拆分成多个子规则,增加了 Bloom filter 命中的概率.所以,SABFE 在实际系统中的性能会更好.

3.2.2 硬件资源比较

基于硬件的算法可能受到硬件的约束以及对于规则集的大小可扩展性限制,文献[14]中基于 KMP 的算法延迟会随着特征字符串的数目成比例地增长,为了增加性能,必须线性增加硬件资源.文献[10]使用 TCAM,成本昂贵,功耗很大.而文献[15]中基于网络处理器的算法依赖于网络处理器强大处理能力、多个并行的执行单元以及昂贵的多端口快速 RAM,而基于 Bloom filter 引擎的算法本身硬件资源开销小[8],实现简单.

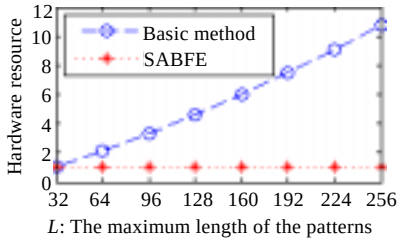

Fig.8 Hardware consumption

图 8 硬件开销

SABFE 与基本的 Bloom filter 引擎方法相比解决了特征字符串长度增加时的可扩展性问题.从图 8 中关于和基本 Bloom filter 引擎硬件资源使用情况的比较中可以看出:基本的 Bloom filter 引擎方法的硬件开销随着规则长度的增加而超线性增加;而 SABFE 随着字符串长度的增加,硬件开销是固定的,具有可扩展性.

这是因为我们只需要一个最大长度为 L_{max} 的 Bloom filter 引擎;而基本的 Bloom filter 引擎方法中基于 Bloom filter 的方案需要最大长度为特征字符串长度的 Bloom filter 引擎.如果要检查最大长度为 L 的特征字符串,基本方法需要 L 个 Bloom filter,在 Bloom filter 引擎上的逻辑开销至少是我们方法的 L/L_{max} 倍.同时,较长的 Bloom filter 的硬件逻辑数目开销还会更大[8];而 SABFE 则有效地克服了这一缺点.

4 结束语

本文总结了深度流搜索的关键点,提出了一种基于有状态 Bloom filter 引擎的高速分组检测算法 SABFE,将状态机的思想引入到使用 Bloom filter 进行分组检测中,构造了快速查找表保存状态机,同时利用前缀寄存器堆保存当前匹配的中间状态信息,从而解决了对于长特征串检测的可扩展性的问题.另外,通过并行查找 Bloom filter 和前缀寄存器堆,以及利用多个并行的 Bloom filter 引擎完成流并行检测,达到了高的吞吐性能,满足了线速的需求.本文对整个系统的吞吐性能和硬件资源消耗进行了详细的分析,证明了 SABFE 确实具有可扩展性和高吞吐量.

References:

[1] Moore D, Paxson V, Savage S, Shannon C, Staniford S, Weaver N. Inside the slammer worm. IEEE Security and Privacy, 2003, 1(4):33−39.

[2] Moore D, Shannon C. Code-Red: A case study on the spread and victims of an Internet worm. In: Proc. of the 2002 ACM SICGOMM Internet Measurement Workshop. Marseille, 2002. 273−284. http://portal.acm.org/citation.cfm?id=637244&dl=ACM&coll=&CFID=15151515&CFTOKEN=6184618

[3] Kim HA, Karp B. Autograph: Toward automatic distributed worm signature detection. In: Proc. of the USENIX Security Symp. Diego, 2004. 271−286. http://www.usenix.org/events/sec04/tech/full_papers/kim/kim.pdf

[4] Singh S, EstanC, Varghese G, Savage S. Automated worm fingerprinting. In: Proc. of the 6th ACM/USENIX Symp. on Operating System Design and Implementation (OSDI). San Francisco, 2004. 45−60. http://www.usenix.org/events/osdi04/tech/full_papers/singh/singh.pdf

[5] Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report, 99-15, Chalmers University, 2000.

[6] Bloom B. Space/Time trade-offs in Hash coding with allowable errors. Communications of the ACM, 1970,13(7):422−426.

[7] Dharmapurikar S, Krishnamurthy P, Sproull T, Lockwood J. Deep packet inspection using parallel Bloom filters. In: Proc. of the Symp. on High Performance Interconnects (HotI). Stanford, 2003. 44−51. http://www.hoti.org/archive/Hoti11_program/papers/hoti11_07_dharmapurikar_s.pdf

[8] Dharmapurikar S, Attig M, Lockwood J. Design and implementation of a string matching system for network intrusion detection using FPGA-based Bloom filters. Technical Report, WUCSE-2004-12, St. Louis: Washington University, 2004.

[9] Song HY, Dharmapurikar S, Turner J, Lockwood J. Fast hash table lookup using extended Bloom filter: An aid to network processing. In: Proc. of the ACM SIGCOMM 2005. Philadelphia, 2005. 20−26. http://portal.acm.org/citation.cfm?id=1080114&dl=ACM&coll=&CFID=15151515&CFTOKEN=6184618

[10] Yu F, Katz RH, Lakshman TV. Gigabit rate packet pattern-matching using TCAM. In: Proc. of the 12th IEEE Int'l Conf. on Network Protocols (ICNP 2004). Berlin, 2004. 174−183. http://portal.acm.org/citation.cfm?id=1025890&dl=GUIDE&coll=GUIDE

[11] Lakshminarayanan K, Rangarajan A, Venkatachary S. Algorithms for advanced packet classification with ternary CAMs. In: Proc. of the ACM SIGCOMM 2005. Philadelphia, 2005. 193−204. http://portal.acm.org/citation.cfm?id=1080115&dl=ACM&coll=&CFID=15151515&CFTOKEN=6184618#

[12] Taylor DE. Survey and taxonomy of packet classification techniques. Technical Report, WUCSE-2004-24, St. Louis: Washington University, 2004.

[13] Baeza-Yates R. Algorithms for string searching: A survey. SIGIR Forum, 1989,23(3-4):34−58.

[14] Bakerand ZK, Prasanna VK. Time and area effificient pattern matching on FPGAs. In: Proc. of the 2004 ACM/SIGDA 12th Int'l Symp. on Field Programmable Gate Arrays (FPGA 2004). Monterey, 2004. 223−232. http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=968312#

[15] Tuck N, Sherwood T, Calder B, Varghese G. Deterministic memory-efficient string matching algorithms for intrusion detection. In: Proc. of the IEEE Infocom Conf. Hong Kong, 2004. 333−340. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1354682

附 录

定理 **1**. SABFE 中长度为 N 的子串的出现概率为 $\sum_{r\in S}\left\lceil\frac{l_r}{N}\right\rceil\left(p_r+\frac{1}{|\varepsilon|^N}\right)$.

证明:Bloom filter 最大可以匹配的规则的长度是 N,那么,对于一条长度为 $l(l\geq N)$的规则来说,SABFE 会拆分

为$\left\lceil \frac{l}{N} \right\rceil$条子串.如果该条规则在流中出现的概率为 P_r,则在拆分的子串的出现概率为 $P_1=\left\lceil \frac{l}{N} \right\rceil p_r$.

假设内容是随机均匀分布的,如果待检测的字符集是ε,集合大小是$|\varepsilon|$,检测流的长度为 L.那么,流中不存在该条规则的匹配,但是存在拆分后的特征子串的概率为 $P_2=\frac{(L-N+1)\times\left\lceil \frac{l}{N} \right\rceil\times\frac{1}{|\varepsilon|^N}}{L}\approx\frac{\left\lceil \frac{l}{N} \right\rceil}{|\varepsilon|^N}(l\geq N)$.

所以,SABFE 对于规则 r 拆分出来的子串的命中概率为 $P_t=p_1+p_2=\left\lceil \frac{l_r}{N} \right\rceil\left(p_r+\frac{1}{|\varepsilon|^N}\right)$.对于规则集 S 来说,将所有属于它的规则对应子串的命中概率相加,得到 SABFE 中长度为 N 的子串出现的概率为

$$\sum_{r\in S}\left\lceil \frac{l_r}{N} \right\rceil\left(p_r+\frac{1}{|\varepsilon|^N}\right).$$ □

定理 **2**. 后端系统分析需要的时间为$\left(d_c+d_u+\left\lceil \frac{8N+2\lceil\log\alpha M\rceil+\lceil\log M\rceil+k\lceil\log N\rceil}{W} \right\rceil\right)\times\frac{1}{F}$.

证明:设内存位宽为 W(bit),Bloom filter 最大匹配长度为 N,硬件的时钟频率为 F.

那么,后端系统分析需要的时间分为如下 3 个部分:

- 读取外存中的表项需要的 t_r 个周期;
- 硬件逻辑的控制时延,d_c 个周期;
- 更新前缀寄存器堆 d_u 个时钟周期.

外存中表项的数据由如下 4 项组成:

- 读取需要精确匹配的子串,N 个 byte,也就是 $8N$ 个 bit;
- 匹配字符串编号必须使用$\lceil\log M\rceil$个 bit 编号,M 是原始规则集的大小;
- 已经命中的前缀编号和构成新前缀的编号必须使用$\lceil\log\alpha M\rceil$个 bit 编号,α是划分特征字符串之后的规则集大小;
- 对于新前缀和可能的后缀之间的距离,必须使用$\lceil\log N\rceil$个 bit 编号,这一项最多出现 k 项.

所以,$t_r=\left\lceil \frac{8N+2\lceil\log\alpha M\rceil+\lceil\log M\rceil+k\lceil\log N\rceil}{W} \right\rceil$个时钟周期,所以,后端系统分析需要的时间为

$$t'=(d_c+d_u+t_r)\times 1/F=\left(d_c+d_u+\left\lceil \frac{8N+2\lceil\log\alpha M\rceil+\lceil\log M\rceil+k\lceil\log N\rceil}{W} \right\rceil\right)\times\frac{1}{F}.$$ □

叶明江(1982－),男,湖南怀化人,博士生,主要研究领域为网络安全,计算机网络体系结构,P2P 覆盖网络.

徐恪(1974－),男,博士,副教授,主要研究领域为新一代互联网体系结构,交换机和路由器体系结构,P2P 与 Overlay 网络.

崔勇(1976－),男,博士,助理研究员,主要研究领域为计算机网络体系结构,协议的仿真和测试,多目标优化的路由算法及其评价.

吴建平(1953－),男,教授,博士生导师,CCF 高级会员,主要研究领域为计算机网络体系结构,计算机网络协议测试.