第 42 卷 第 6A 期
2015 年 6 月

计 算 机 科 学
Computer Science

Vol. 42 No. 6A
June 2015

基于布隆过滤器的精确匹配算法设计与实现

王鹏超 杜慧敏 曹广界 杜琴琴 丁家隆

(西安邮电大学电子工程学院 西安 710061)

摘 要 针对布隆过滤器技术存在将不属于该集合的某元素误判为属于该数据集合(假阳性)和元素删除困难的问题,提出了 CAM(内容可寻址存储器)来进行二级匹配。与直接将字符串存储在 CAM 的单级匹配模式不同,提出将布隆过滤器的 k 个哈希值存入 CAM,从而判断某元素是否真正属于这个集合,从而达到精确匹配,且易于删除元素。对算法在 Snort2.9 规则库下的分析结果表明,相较于单级 CAM 查找,所设计的两级匹配模式在假阳率为 0.01 时,系统的资源占用减少 5 倍以上;本算法功耗降低 10 倍以上,能够减轻系统的负载,提高系统性能,适用于高速网络中字符串的检测。

关键词 布隆过滤器,内容寻址存储器,字符串匹配,哈希函数,网络安全

中图法分类号 TP338.6 文献标识码 A

Design and Implement of Exact Matching Algorithm Based on Bloom Filter

WANG Peng-chao DU Hui-min CAO Guang-jie DU Qin-qin DING Jia-long

(School of Electronic Engineering, Xi'an University of Posts & Telecommunications, Xi'an 710061, China)

Abstract Because the technology of Bloom filter makes the element which does not belong to a set misjudge the element belonging to the set and element removal is difficult, this paper introduced CAM(Content Addressable Memory) for two exact matches. We proposed that the k hash values of the Bloom filter store in CAM, in order to determine whether an element belongs to this collection, and it is easy to remove elements. The results of the algorithm based on Snort2.9 rule database show that, compared with single-stage CAM seeking, in false positive rate of 0.01, the BF-CAM system reduces by more than 5-fold reduction in resource consumption, 10 times in power consumption, and it can reduce the load of system, improve system performance, and adapt to detect strings in the high-speed networks.

Keywords Bloom filter, CAM, String matching, Hash function, Network security

随着 Internet 的快速发展,网络攻击现象呈爆炸式增长,网络安全面临着巨大的挑战,日益得到人们的高度重视。内容安全是网络安全的重要组成部分,正受到越来越多的关注和研究。深度包检测技术[1]是一种先进的包过滤技术,在分析数据包包头的基础上,对数据包的有效载荷进行扫描和检测,能够有效识别出隐藏在数据包有效载荷内的非法数据,目前该技术已经成为网络安全技术中的研究热点。现在绝大多数网络安全产品均是基于攻击库而对网络数据进行模式匹配处理,其匹配性能直接决定了系统的检测时间,进而成为决定系统安全能力的关键。如文献[2]指出,在开源入侵检测系统 Snort[3] 中,字符串匹配已占据整个系统 70% 以上的处理时间。因此,开发出高速硬件字符串匹配检测系统迫在眉睫。

由于 Bloom Filter 是一种空间效率极高的随机数据结构,其内部运算只需简单的"与"和"异或"操作,易于硬件实现,且匹配效率极高,已广泛应用于网络安全检测系统中。但是 Bloom Filter 存在将不属于集合的某元素误判为属于该数据集合(假阳性)和元素删除困难的问题。针对该问题,本文提出了采用 CAM(内容可寻址存储器)进行二级精确匹配,从而判断该元素是否真正属于这个集合且易于删除元素。Bloom Filter 和 CAM 的两级匹配模式,利用了 Bloom Filter 处理速度快的特点,首先将数据流中的大部分正常数据快速过滤掉,能够减轻整个系统的负载,然后利用 CAM 精确匹配的优点,可以极大提高系统的匹配速度和精确度。

1 相关工作

自从 Bloom Filter 提出后,后人在 Bloom Filter 的假阳率、存储、元素删除等方面进行了深入研究。

标准的 Bloom Filter 仅支持元素的插入与查找,不支持元素删除,Counting Bloom Filter[4] 的出现解决了这个问题。Counting Bloom Filter 将标准 Bloom Filter 位数组的每一位扩展为一个小的计数器,在插入元素时给相应的 Counter 值加 1,删除元素时给对应的 Counter 值减 1。Counting Bloom Filter 虽然解决了元素删除,但却增加了约 4 倍的存储空间代价。

Bloom Filter 常用于网络中信息交换传递的信息,因此我们希望消息在传递之前能够被压缩。Compressed Bloom Fil-

本文受国家自然科学基金(90607008,60976020),陕西省政府基金(2011k06-47)资助。

王鹏超(1990—),男,硕士,主要研究方向为电路与系统,E-mail:529962391@qq.com;杜慧敏(1966—),女,博士,教授,主要研究方向为集成电路设计、计算机体系结构;曹广界(1990—),男,硕士,主要研究方向为集成电路系统设计;杜琴琴(1989—),女,硕士,主要研究方向为计算机体系结构与 VLSI 设计;丁家隆(1991—),男,硕士,主要研究方向为电路与系统。

ter[5]的提出减少了存储,降低了假阳率,但是却增加了消息的压缩和解压缩的代价。

Partial Bloom Filter[6]和标准 Bloom Filter 唯一不同的是哈希函数的映射范围。在 Partial Bloom Filter 中,位数组被等分成 k 个区域,每个哈希函数只映射到其中一个区域。Partial Bloom Filter 的假阳率比标准 Bloom Filter 稍大一点。在实际应用中,Partial Bloom Filter 有一定的优势,因为一旦哈希函数的映射范围也独立开来,k 个哈希函数就可以并行访问位数组,从而提高系统性能。

2 Bloom Filter 概述

2.1 Bloom Filter 基本概念

Bloom 算法[7]是由 Burton Bloom 在 1970 年提出的随机数据存储结构,目前已被广泛地应用于各种计算机系统之中,用于表达庞大的数据集及提高数据查找效率。

Bloom Filter[8]是一种空间效率极高的随机数据结构,利用位向量很简洁地表示一个集合,并能够判断一个元素是否属于该集合。Bloom filter 是一种基于多哈希(hash)函数映射压缩参数空间的数据结构,将待匹配数据同 k 个 hash 函数运算后映射到一个初始化为全 0 的 m 维位向量中,相比于传统的比较模式而言,大幅减小了匹配存储空间及比较时间。

Bloom Filter 匹配的结构如图 1 所示,主要包含 3 部分:hash 函数、m 维位向量及匹配结果处理。

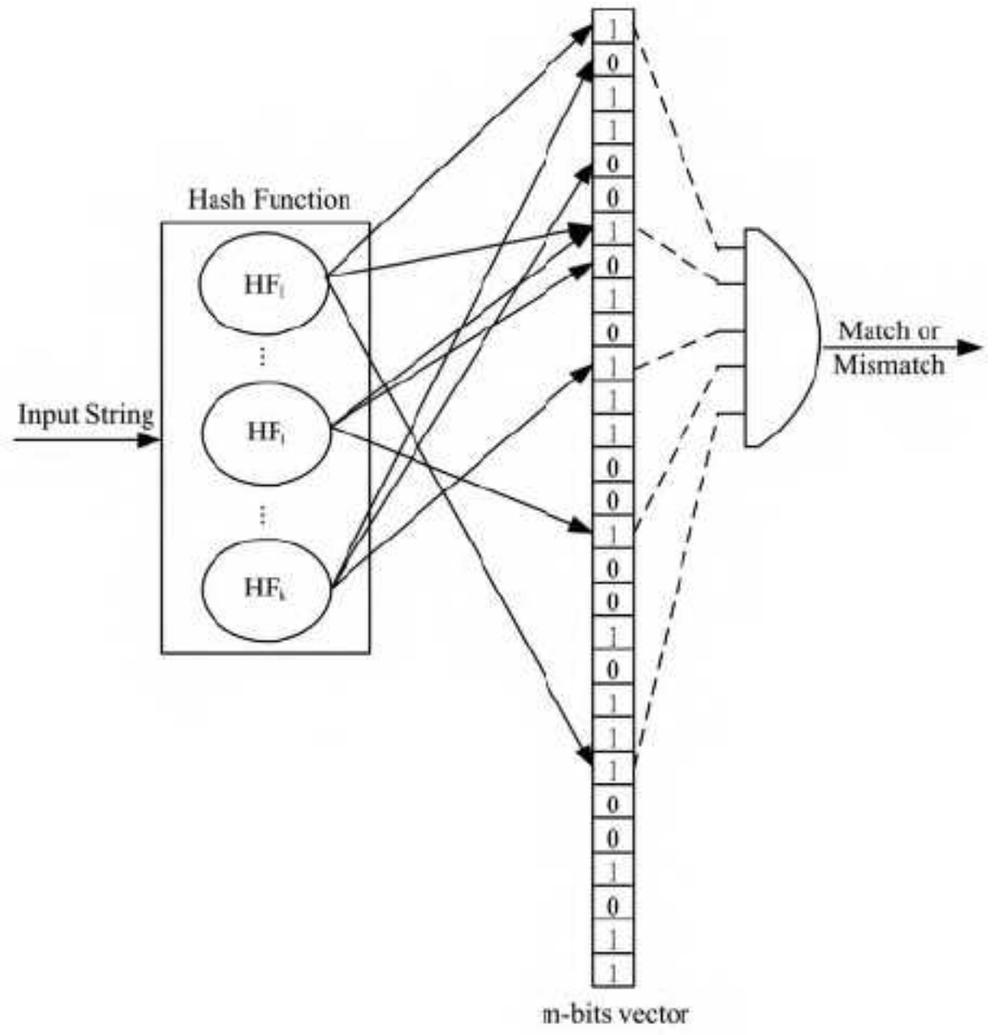

图 1 Bloom Filter 匹配结构

2.2 Bloom Filter 工作原理

Bloom filter 的具体工作原理是:首先定义长度为 m 的位向量 M,并设计哈希函数集 $H=\{h_1,h_2,\cdots,h_k\}$ 及规则集 $R=\{r_1,r_2,\cdots,r_n\}(1\leqslant i\leqslant n)$。

其次将规则集 R 中的所有元素写入 Bloom Filter 中,步骤为:

1. 对位向量 M 进行初始化,并将其 m 位全置为 0;

2. 对元素 r_i,将其同 k 个哈希函数运算,得到 k 个值域为 $[0,m-1]$ 的哈希值 $h_1(r_i),h_2(r_i),\cdots,h_k(r_i)$;

3. 检查数组 M 中的相应位 $m[h_1(r_i)],m[h_2(r_i)],\cdots,m[h_k(r_i)]$ 的值,如该位值为 0,则将 M 中该位置为 1;若该位值为 1 则不做任何操作;

4. 重复步骤 2 和 3,直到位于规则集 R 中的所有元素都写入到 M 中。

对输入数据进行查询时,将其同 k 个哈希函数运算得到 k 个哈希值,然后检查数组 M 中的相应位 $m[h_1(r_i)],m[h_2(r_i)],\cdots,m[h_k(r_i)]$ 的值是否全为 1,若有一位为 0,则该输入数据肯定不属于规则集;若全为 1,则以一定的误判率判定该输入数据属于规则集。

2.3 Bloom Filter 的假阳性

对 Bloom Filter 的假阳性以下面例子说明。

首先定义 Bloom Filter 位向量 M 的长度为 10,哈希函数集 $H=\{h_1,h_2,h_3\}$,规则集 $R=\{r_1,r_2\}$,如图 2 所示。

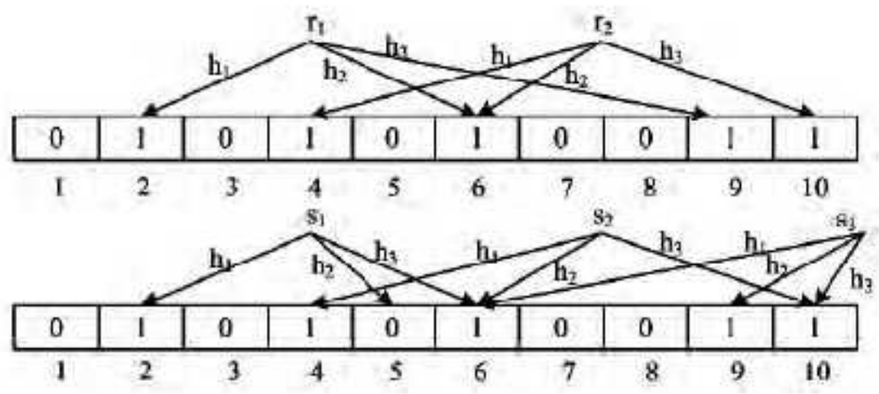

图 2 Bloom Filter 假阳性分析图

元素 r_1 经过 3 个哈希函数得到相应的 3 个哈希值:$h_1(r_1)=2,h_2(r_1)=6,h_3(r_1)=9$,现将 $m[2],m[6]$ 和 $m[9]$ 的值全置为 1,同样,将元素 r_2 经过哈希运算后的 $m[4],m[6]$ 和 $m[10]$ 位置 1,当将 r_2 写入 $m[6]$ 时,检查 $m[6]$ 的值后发现 $m[6]=1$,于是不再重复将 $m[6]$ 的值置为 1 的操作。

设字符串 s_1、s_2 和 s_3 待匹配,同样利用上述 3 个哈希函数得到各自 3 个哈希值,s_1 经过哈希函数得到值为 2、5、6,查询得 $m[2]=1,m[5]=0,m[6]=1$,由此可判定元素 s_1 不属于规则集 R。s_2 经过哈希函数得到值为 4、6、10,查询得 $m[4]=m[6]=m[10]=1$,由此可判定元素 s_2 不属于规则集 R。对于字符串 s_3,经过哈希函数得到值为 6,9,10,查询得 $m[6]=m[9]=m[10]=1$,于是可判断元素 s_3 属于规则集 R。但由上文可知 $m[6]=m[9]=1$ 是 r_1 的存在引起的,且 $m[10]=1$ 是因为有 r_2 存在的缘故,因此可得其实判定元素 s_3 属于规则集 R 是一个误判。

下面对 Bloom Filter 的假阳性进行理论分析。

假设位向量 M 的长度为 m,哈希函数的个数为 k,规则集中的元素个数为 n,那么当某个元素同 k 个哈希函数进行运算后加入长为 m 的位向量时,位向量中某位是 0 的概率 p_0 为 $p_0=(1-\frac{1}{m})^k$。规则集 R 中所有元素都运算得到哈希值且对应到 M 后,M 某一位仍为 0 的概率 p 为 $p=(1-\frac{1}{m})^{kn}$。

由方程 $\lim\limits_{x\to\infty}(1-\frac{1}{x})^{-x}=e$ 可得出 p 值近似解 $p=e^{(-kn/m)}$,因此假阳性发生的概率[7,9] $f=(1-p)^k=(1-e^{-kn/m})^k$。

针对 Bloom Filter 假阳性,本文设计提出采用 CAM 进行二级精确匹配,下面对 CAM 进行介绍。

3 CAM 介绍

3.1 CAM 基本概念

CAM 是一种新型的存储器,它的高速、并行、易扩展和实现的灵活性使它一出现就得到了人们的重视[11]。CAM 是基

于内容寻址的，通过硬件电路实现快速匹配，广泛应用于以太网网址搜寻、数据压缩、模式识别、高速缓存、高速数据处理、数据安全和数据加密等[12,14]。

3.2 CAM 工作原理

CAM 是一种专门为快速查找数据地址而设计的存储器，通过把输入数据与其内所存数据相比较，能较快速确定输入数据是否与其内部某个数据或某几个数据相匹配。CAM 的数据寻址方式因应用要求的不同而不同，最快的寻址方式下，仅需要一个时钟周期就可完成对所有数据的寻址[12-14]。

一个 CAM 的核心部分是由 CAM 单元构成的阵列，典型的 CAM 基本单元结构如图 3 所示。可以看出，其存储单元主要是标准的 6 管静态 SRAM 单元，CAM 单元就是在 SRAM 单元的基础上增加了相应的比较器，可用于实现快速的搜索、匹配操作。

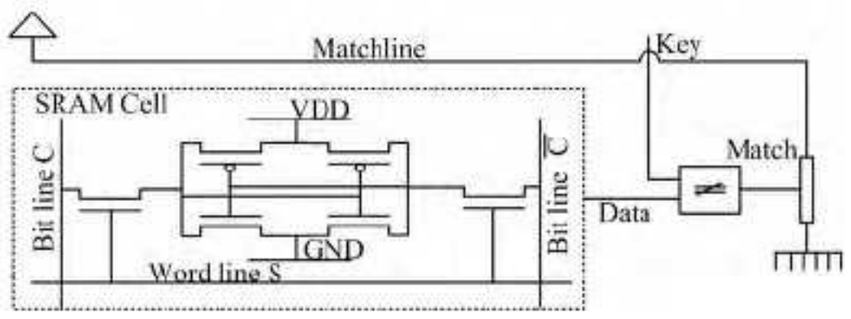

图 3 典型的 CAM 单元结构

CAM 采用阵列式数据存储，读取模式如图 4 所示。

图 4 CAM 读取模式

在 CAM 中，输入的是所要查询的数据，输出的是数据所在地址和匹配标志，若匹配，则输出数据地址。如对于图 4 中的 CAM，其内部存储单元有 1024 个，每个单元的位宽为 8，那么当输入待查询数据时，将在 CAM 的 1024 个单元中同时查找，若数据存在，则匹配标志 Match 为高，并得到数据所在地址。由于 CAM 不是采用传统的地址线模式读取数据，因此存储空间可以很容易地扩展，输入数据宽度只由所查询的数据位数决定。

4 BF-CAM 算法

为叙述方便，下文将 Bloom Filter 简称为 BF。

4.1 算法概述

BF-CAM 算法采用 BF 和 CAM 的两级匹配模式，利用 BF 将数据流中的大部分正常数据过滤掉，以减轻 CAM 的匹配压力，通过 CAM 排除假阳性，达到精确匹配。

BF-CAM 算法的基本思想是：

(1) 将待匹配数据经过第一级 BF，如果没有匹配，则该数据不属于这个规则集；若匹配，则进入(2)；

(2) 对于第一级处理后的数据进行 CAM 精确匹配，如果匹配，则该数据属于这个规则集，否则不属于该规则集。

4.2 举例说明

4.2.1 存储

1. BF 的存储

BF 中的各参数以 2.3 节中例子进行说明。

BF 的存储主要是针对其 m 维位向量的，将规则集中的元素同哈希函数进行运算后，把得到的哈希值写入 m 维位向量中。在该例子中，将 m 维位向量中 2、4、6、9、10 单元置为 1。

2. CAM 的存储

将经过 BF 中哈希函数得到的 k 个哈希值存储到 CAM 中，即 $\{h_1(r_i), h_2(r_i), \cdots, h_k(r_i)\}$。对应该例子中，即将 $\{2,6,9\}$ 和 $\{4,6,10\}$ 写入 CAM 中。

另外，传统的 BF 不支持删除元素功能，添加了二级 CAM 后，可以支持删除操作。如若将规则集中的 r_2 删除，那么只需将 CAM 中的 $\{4,6,10\}$ 单元删除即可。

4.2.2 匹配

通过前面 2.3 节分析可知，s_1 肯定不属于该规则集 R，s_2 和 s_3 以一定的误判率属于该规则集，需经过第二级 CAM 进行精确匹配。s_2 经过哈希函数后得到的 3 个哈希值为 $\{4,6,10\}$，将 s_2 通过 CAM 查询时，该值存在于 CAM 中，因此断定 s_2 确实属于该规则集。同样，对于 s_3，将其对应 3 个哈希值 $\{6,9,10\}$ 输入至 CAM 进行查询，查询可得该值在 CAM 中不存在，因此可以得到 s_3 不属于该规则集。这样，就能将假阳性的元素排除，得到精确的匹配结果。

5 BF-CAM 算法的硬件结构

5.1 硬件结构

BF-CAM 算法的硬件结构如图 5 所示。下面对各个模块的功能进行描述。

Bloom Filter 模块：主要包含 hash、匹配向量和 BF 结果仲裁模块。hash 模块将输入数据进行哈希运算，得到哈希值；写入时，匹配向量将哈希值所对应位置为 1，查询时，查询对应位置的值；BF 结果仲裁模块用于查询前一模块得到的哈希值是否为全 1，从而得出 BF 查询的匹配结果；

CAM 模块：主要是将 BF 匹配成功的数据进行二次查表，得到 CAM 的匹配结果；

结果处理模块：根据 BF 和 CAM 的结果仲裁，得到该数据的最终匹配信息。

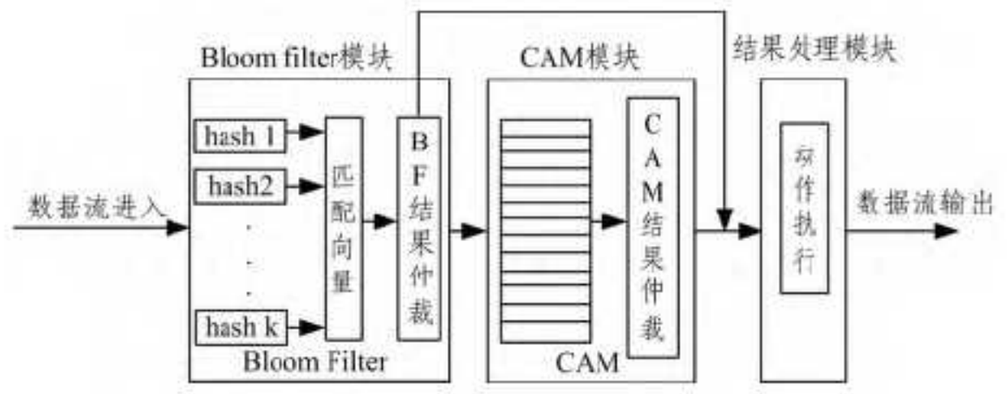

图 5 BF-CAM 硬件结构

5.2 相关参数的构建

1. 哈希函数的选择

对于哈希函数的选择，随机性和散列性越强，硬件设计的效果越好。由于 H3 哈希函数具有很强的散列性，是一种常见的布隆过滤器的实现函数，又因其对于每个输入元素的哈希计算仅需要简单的“与”和“异或”运算，便于实现，尤其是硬件实现，因此本设计采用的哈希函数为 H3 哈希函数。

H3 哈希函数实际上是一个线性转换 $B_{r\times1}^T = Q_{r\times w} \cdot A_{w\times1}^T$，将 w-bit 长度的元素 $A = a_1 a_2 \cdots a_w$ 转化为 r-bit 的哈希地址 $B = b_1 b_2 \cdots b_r$（$r < w$，将较大长度的元素运算后变短，使设计简化），即

$$\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_r \end{pmatrix} = \begin{pmatrix} q_{11} & q_{12} & \cdots & q_{1w} \\ q_{21} & q_{22} & \cdots & q_{2w} \\ \cdots & \cdots & \cdots & \cdots \\ q_{r1} & q_{r2} & \cdots & q_{rw} \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_w \end{pmatrix}$$

其中，$Q_{r\times w}$ 矩阵中的每一个元素 q_{ij} 都是随机的 0 和 1，设计中的乘法运算和加法运算分别采用二进制与(AND)和二进制异或(XOR)运算，即 $b_i=(a_1\cdot q_{i1})\oplus(a_2\cdot q_{i2})\oplus\cdots\oplus(a_w\cdot q_{iw})$，$i=1,2,\cdots,r$。

2. 哈希函数个数 k 以及位数组 m 的计算

假设全集中共有 n 个元素，允许的最大错误率为 p，位数组的位数为 m，哈希函数个数为 k。

首先由插入到过滤器中的元素数 n 和希望的误差率 p，得到最优 k 和 m 的值，计算方法为 $k=0.7\cdot\frac{m}{n}$，$m=-\frac{n\cdot\ln p}{(\ln 2)^2}$[9,10]。因此，在哈希函数的个数取到最优时，要让错误率不超过 p，m 至少需要取到最小值 $n\log_2\frac{1}{p}$ 的 1.44 倍，假设 $p=10^{-a}$，则 $m=4.8\times a\times n$，$\frac{m}{n}=4.8a$；可推得 $k=3.36a$。故哈希函数个数只与错误率有关，若假阳性概率为 0.001，即 $a=3$，那么哈希函数的个数为 $k=3.36a=10$，且 $m=\lfloor 14.4n\rfloor$。

3. m 位数组的构建

对于 m 位数组的构建，采用 RAM 来实现，其大小为 $m\times 1$。例如对于 BF2，m 为 4，则构建一个 4×1 的 RAM，并将其初始化为全 0，将哈希运算后的哈希值所对应的表项置为 1。

4. Q 矩阵的构建

对线性变换 Q 矩阵的构建，也采用 RAM 实现。由于 Q 矩阵中的每一个元素都是随机的 0 和 1，采用伪随机数产生器产生伪随机数，将其注入到 RAM 中。例如需要一个 $Q_{2\times 16}$ 的矩阵，产生 32 位的伪随机数，将其作为一个 2×16 的矩阵，即可构建 Q 矩阵。

$Q_{r\times w}$ 矩阵的参数：由于 $B^{\mathrm{T}}_{r\times 1}=Q_{r\times w}\cdot A^{\mathrm{T}}_{w\times 1}$，而 $m=4.8\times a\times n$，且 $m=2^r$(因为 r 根地址线可以确定 m 的大小)，因此 $r=\log_2 m$。对于 i 个字节，w 为 $8i$，故 Q 矩阵的参数为 $Q_{(\log_2 m\times 8i)}$。

6 多模式的 BF-CAM 引擎

定义 1(多模式的 BF-CAM) 其指的是系统每次不仅仅处理一个定长的字符串，还可以处理变长字符串。

对于网络中连续的数据流，如何能够实现连续检测不同长度的字符串，因此，本文采用多模式的 BF-CAM 算法。

若每个时钟周期进入系统 1 个字节，图 6 是详细的单字节多模式 BF-CAM 过滤引擎原理示意图，简称 MMSB。该引擎由一组特征字符串长度从 $L_{\min}$ 到 $L_{\max}$ 之间组成。在图 5 中，$L_{\min}$ 为 3 字节，$L_{\max}$ 为 w 字节。被检测字符串每次移动一个字节，所有 BF-CAM 并行工作，这样单字节多模式的 BF-CAM 引擎就可以将输入的字符串和规则集进行完全匹配。

若每个时钟周期进入系统的字节数大于 1，则相应的多字节多模式 BF-CAM 引擎(简称 MMMB)如图 7 所示。

假设每次输入的字节数为 n，图 7 所示的多字节多模式 BF-CAM 引擎主要由 n 个单字节多模式 BF-CAM 过滤引擎组成，检测窗口每次移动 n 个字节。由于每组 MMSB 所存储的规则相同，因此在系统内部设计了共享存储，让 n 组的 MMSB 共享一组存储表。

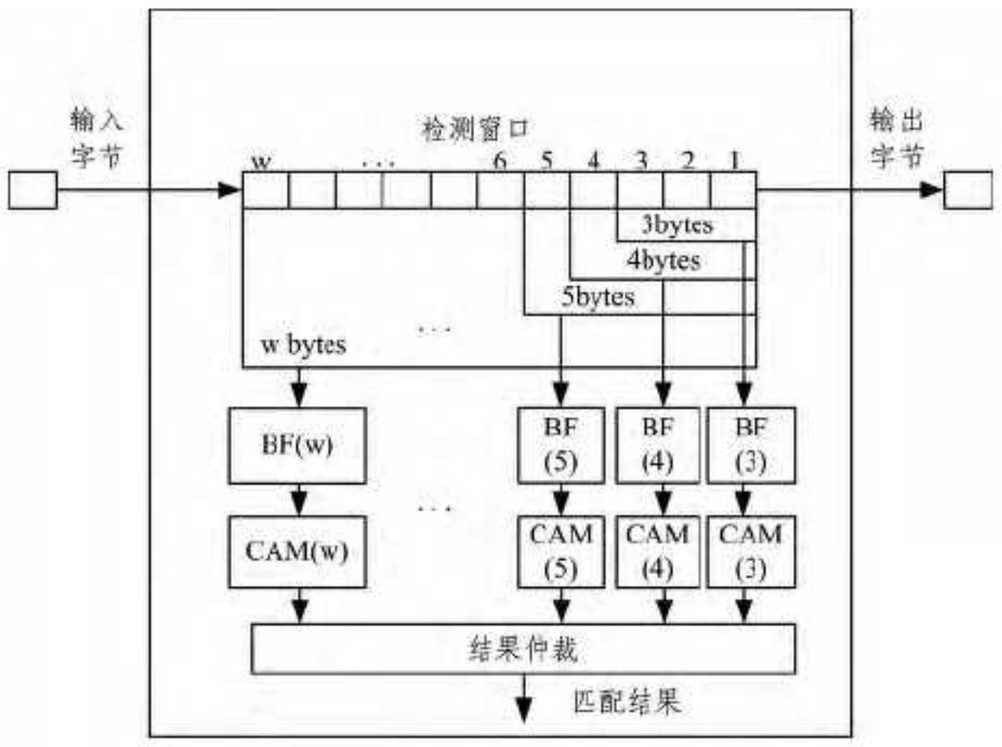

图 6 单字节多模式 BF-CAM 引擎

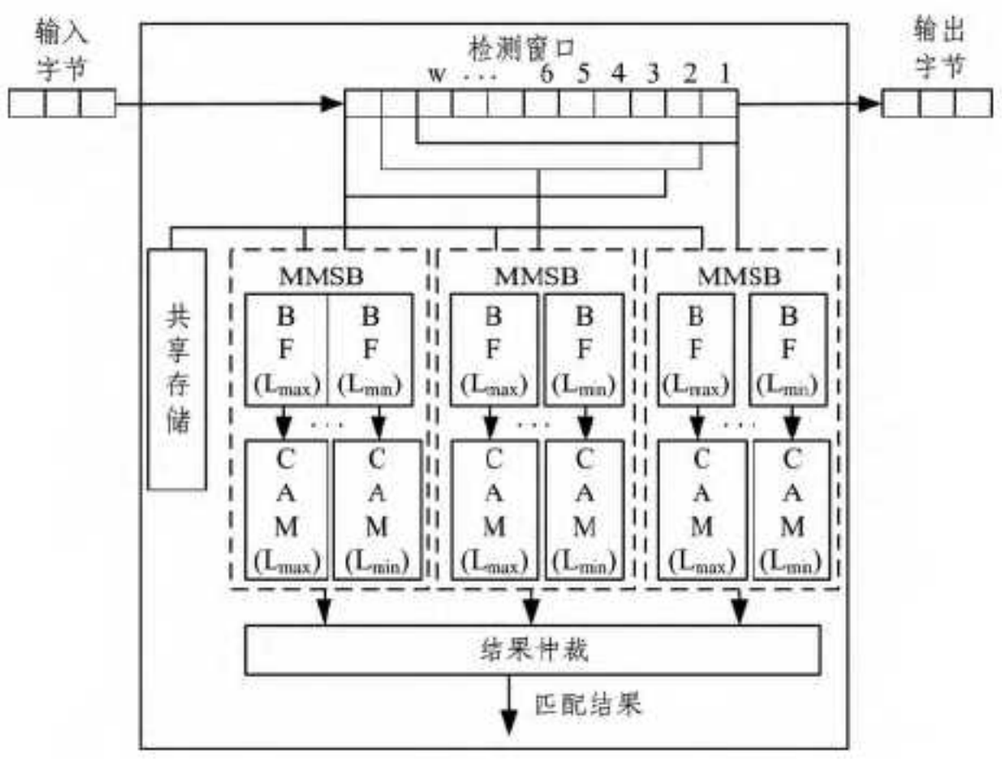

图 7 多字节多模式 BF-CAM 引擎

7 实验及性能分析

由于 BF 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，因此一级匹配能够达到很高的性能。而二级的 CAM 采用阵列式数据存储，能够对内部数据进行并行检测，匹配效率也非常高。

7.1 BF-CAM 性能分析

实验采用的硬件平台为 Xilinx ISE 12.1，基于 Xilinx 系列 Virtex5(型号为 xc5vlx220，时钟频率为 200MHz)的 FPGA 器件进行模拟仿真。采用 Snort2.9 规则库[17]数据对存储信息进行分析，由于 Snort2.9 规则的特征字长度在 40 以内占 97%，因此本文对长度在 40 以内的特征字进行分析，特征字长度统计如图 8 所示。

文献[16]提出了将字符串直接存储到 CAM 的方法进行查找匹配，能提高系统匹配效率且可实现精确匹配，下面将本文所设计的 BF-CAM 与文献[16]的算法进行对比。

文献[17]指出，CAM 所占存储大小与所存储数据的宽度和深度的乘积成线性关系。接下来分析 BF-CAM 中 CAM 所占的存储大小。

对于 BF-CAM 架构中 CAM 的存储，CAM 的宽度 L 为：哈希函数的个数 $\times$ 每个哈希函数值的宽度，即 $L=k\times\log_2 m$，单位为 bit。由 5.2 节可知，k 只与错误率有关，而 $m=\lfloor 14.4n\rfloor$，

则 $L=k\times\log_2|14.4n|$，因此总共所需的存储为 $Mem_1=n\times L=n\times k\times\log_2|14.4n|$，且 Mem_1 与 n 和误码率 p 有关。

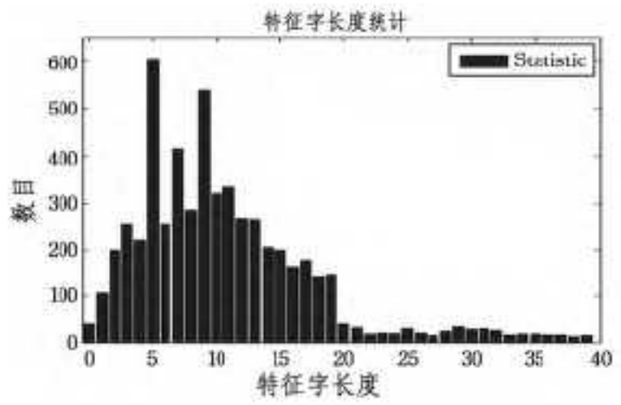

图 8　Snort2.9 规则的特征字长度统计

文献[16]采用将字符串直接存储到 CAM 的方法进行查找，那么 CAM 所需要的存储为 $Mem_2=n\times 8\times i$（i 为特征字的长度，8 表示一个字符占 8bits），因此，Mem_2 与 n 和 i 有关。

文献[15]指出，常用 BF 过滤器的误码率一般为 0.01，因此在 $p=0.01$ 情况下对 Mem_1 和 Mem_2 进行比较。

当 $a=2$ 时，$p=10^{-a}=0.01$，此时 $Mem_1=n\times 6\times\log_2|14.4n|$，$Mem_2=n\times 8\times i$。

由图 9 可知，在误码率 $p=0.01$ 时，特征字长度在 1 到 6 时，两者存储相差不大，但当特征字长度在 7 到 40 时，$Mem2$ 的存储相对于 $Mem1$ 能够减少 5 倍以上，因此 BF-CAM 算法中 CAM 的存储开销较小。

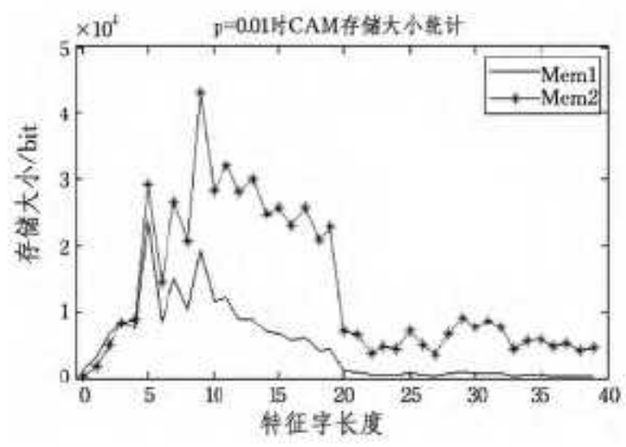

图 9　$p=0.01$ 时 CAM 存储大小统计

由于 BF 的存储资源主要为 m 维位向量的大小，而位向量的宽度只有 1 位，因此其存储开销相对较小。通过对 BF-CAM 电路进行综合，得到 BF 的资源消耗相对于 CAM 来说是非常少的，这是由 CAM 的并行查找机制所决定的，因此 BF-CAM 的主要存储为内部 CAM 的开销。使用 Xilinx 功耗估计器(XPS)[18]分析了 CAM 和误码率 $p=0.01$ 时 BF-CAM 系统相应的功耗，如图 10 所示。

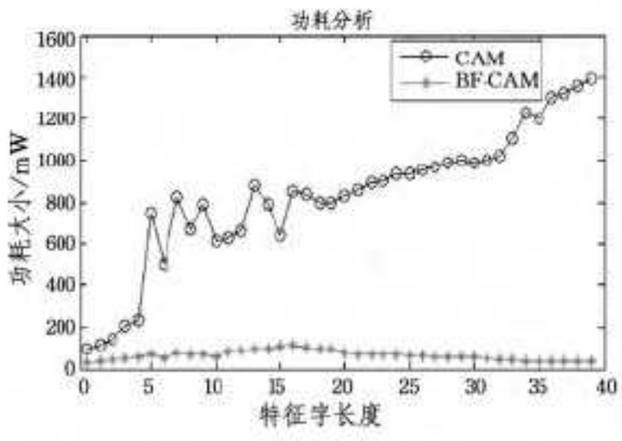

图 10　系统功耗分析

由图 10 功耗情况分析可知，相较于单级 CAM 的功耗，BF-CAM 的功耗是非常小的。这是由于 CAM 的功耗与其宽度和深度成线性比例[17]，在 CAM 深度一定时，CAM 的宽度越大，其功耗越大，且变化的比例系数增加。传统的 CAM 单级匹配，其宽度随着特征字长度的变大而增加，导致特征字长度在 18 字节以上时功耗相当大；BF-CAM 的宽度不随特征字长度的变大而增加，由于其宽度 $L=k\times\log_2 m$，在 $p=0.01$，特征字长度变化时，CAM 的宽度基本在 50bits 上下变动，因此能够达到较低的功耗。

相较于单级的 CAM 匹配，本文提出的 BF-CAM 在 $p=0.01$ 时，功耗能够降低 10 倍以上，是由于 BF-CAM 架构只将少量的数据流通过 CAM，减少了 CAM 的访问次数，降低了整个系统的功耗。因此，本文提出的 BF-CAM 算法相对于 CAM 算法具有较低的存储开销及更小的功耗，适合应用于高速网络中字符串的检测。

7.2　Hash 冲突的说明

H3 哈希函数的转换是一种压缩映射，因此，Hash 冲突的存在也是在所难免的。下面分析本设计的 Hash 冲突情况。

由 5.2 节可知，若规则集大小为 n，错误率 $p=10^{-a}$，则哈希函数个数 k 只与错误率有关，即 $k=3.36a$，位数组 $m=4.8\times a\times n$。如图 11 所示，若两组字符串 c_1 和 c_2 发生 Hash 冲突，则对于 k 个哈希函数来说，将 c_1 和 c_2 要映射到 m 位位向量的相同位置上，即 $h_1(c_1)=h_1(c_2)$、$h_2(c_1)=h_2(c_2)\cdots h_k(c_1)=h_k(c_2)$，下面计算这种情况下发生的概率。

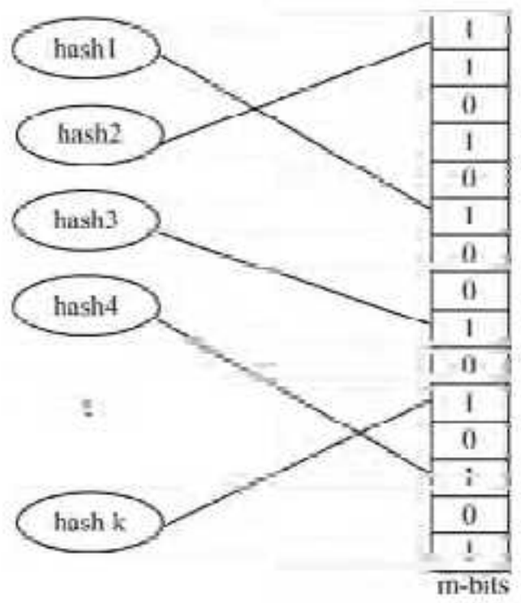

图 11　Hash 冲突分析

若 $h_1(c_1)=t_1$，则 $p(h_1(c_1)=h_1(c_2)=t_1)=\frac{1}{m}$；同理，若 $h_2(c_1)=t_2$，则 $p(h_2(c_1)=h_2(c_2)=t_2)=\frac{1}{m}$；因此 $p(h_1(c_1)=h_1(c_2),h_2(c_1)=h_2(c_2)\cdots h_k(c1)=h_k(c_2))=\frac{1}{m^k}$。下面用 snort 规则库来分析该 Hash 冲突率。

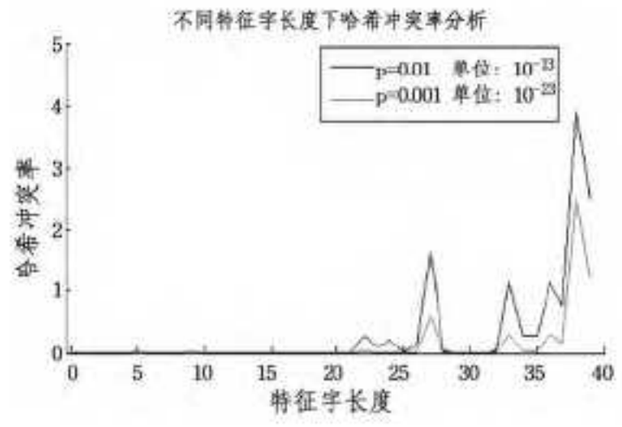

图 12　不同特征字长度下 Hash 冲突分析

图 12 所示为在不同错误率下，不同特征字长度下 Hash 冲突发生概率。在 $p=0.01$ 时发生哈希冲突的概率为 10^{-13} 数量级，在 $p=0.001$ 时发生哈希冲突的概率为 10^{-23} 数量级，该概率非常低，接近于 0。对 snort 规则进行分析后，在实际电路设计中，选择合适的 H3 哈希函数，能够有效地避免哈希冲突的发生，实现精确匹配。

7.3 多模式 BF-CAM 分析

由于网络流量中恶意流量比例 pt 决定了系统工作的检测速度及吞吐性能，因此，实验测试了不同 pt 下得到系统的吞吐性能。如图 13 所示，其中匹配模式最大长度即为 $L_{max}-L_{min}$。

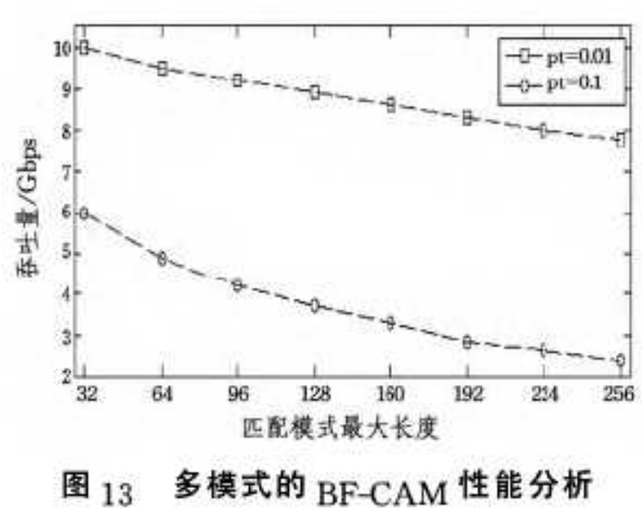

图 13 多模式的 BF-CAM 性能分析

通过实验分析得到：BF-CAM 在 pt 为 1% 时，能够达到 5Gbps 以上的检测速率，当 pt 为 10% 时，系统能够达到 1Gbps 以上的检测速率。另外，BF-CAM 的硬件资源开销较小，可以通过增加并行硬件资源进一步提高系统的性能。

结束语　随着计算机网络的高速发展，网络和信息安全对国家安全、社会稳定的影响越来越大，而当前网络中的安全问题日趋严重，各种网络攻击已经给网络安全造成了极大的威胁。本文在研究基于 Bloom Filter 模式匹配引擎的基础上，提出了采用 Bloom Filter 和 CAM 相结合的设计思想，排除了 Bloom Filter 的假阳性，实现精确匹配以及易于元素的删除。由于 Bloom Filter 匹配速度极快，因此本算法能够减轻系统的负载，大大提高了系统处理速度。在 Xilinx 系列 Virtex5 型号的 FPGA 器件上实验性能分析可得，系统的吞吐量能够达到 Gbps，达到了较高的吞吐性能。且相对于单级的 CAM 查找，本文提出的 BF-CAM 算法能够表达庞大的数据集以及提高查找效率。结论表明，在假阳率为 0.01 时，整个系统的资源占用减少 5 倍以上，功耗可以降低 10 倍以上。下一步工作将根据本文的分析，按照可并行点做进一步的优化，使系统达到更高的性能，满足高速网络的需求。

参 考 文 献

[1] Ruijie Network. DPI Technical Papers[OL]. http://wenku.baidu.com/ iew/aa73eac66137ee06ef f91879. html. 2010. 3

[2] Tuck N, Sherwood T, Calder B, et al. Deterministic memory-efficient string matching algorithms for intrusion detection[C]// Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004). IEEE, 2004, 4: 2628-2639

[3] Introduction to Snort[OL]. http://www. snort. org/docs/

[4] Song H, Dharmapurikar S, Turner J, et al. Fast hash table look-up using extended bloom filter: an aid to network processing[J]. ACM SIGCOMM Computer Communication Review, ACM, 2005, 35(4): 181-192

[5] Mitzenmacher. Compressed bloom filters [J]. IEEE/ACM Transactions on Networking(TON), 2002, 10(5): 604-612

[6] Peir J K, Lai S C, Lu S L, et al. Bloom filtering cache misses for accurate data speculation and prefetching[C] // Proceedings of the 16th International Conference on Supercomputing. ACM, 2002: 189-198

[7] Bloom B H. Space/time trade-offs in hash coding with allowable errors[J]. Communications of the ACM, 1970, 13(7): 422-426

[8] Mitzenmacher M. Compressed bloom filters [J]. IEEE/ACM Transactions on Networking, 2002, 10(5): 604-612

[9] 谢鲲. 布鲁姆过滤器查询算法及其应用研究[D]. 长沙：湖南大学, 2007

[10] Tan J S, Kuang Z, Yang G. Bloom filter based frequent patterns mining over data streams [C] // 2012 International Conference on Graphic and Image Processing. International Society for Optics and Photonics, 2013: 87685V-87685V-7

[11] 徐欣，李宗华，卢启中，等. 基于 FPGA 的内容可寻址存储器研究设计与应用[J]. 国防科技大学学报, 2001, 23(5): 69-73

[12] Guo R, Delgado-Frias J G. IP Routing table compaction and sampling schemes to enhance TCAM cache performance[J]. Journal of Systems Architecture, 2009, 55(1): 61-69

[13] Kim Y D, Ahn H S, Kim S, et al. A high-speed range-matching TCAM for storage-efficient packet classification [J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2009, 56(6): 1221-1230

[14] Chen Y, Oguntoyinbo O. Power efficient packet classification using cascaded bloom filter and off-the-shelf ternary CAM for WDM networks[J]. Computer Communications, 2009, 32(2): 349-356

[15] Kanizo Y, Hay D, Keslassy I. Access-efficient balanced Bloom filters[J]. Computer Communications, 2013, 36(4): 373-385

[16] AbuHmed T, Mohaisen A, Nyang D H. A survey on deep packet inspection for intrusion detection systems[J]. Magazine of korea Telecommunication Society, 2007, 24(11): 25-36

[17] McLaughlin K, O'Connor N, Sezer S. Exploring CAM design for network processing using FPGA technology [C] // International Conference on Internet and Web Applications and Services/ Advanced International Conference on Telecommunications, 2006 (AICT-ICIW'06). IEEE, 2006: 84-84

[18] 田耘，徐文波. Xilinx FPGA 开发实用教程[M]. 北京：清华大学出版社, 2008

(上接第 407 页)

[25] Bertoni G, Zaccaria V, Breveglieri L, et al. AES power attack based on induced cache miss and countermeasure[C] // International Conference on Information Technology: Coding and Computing, 2005(ITCC 2005). IEEE, 2005, 1: 586-591

[26] Acliçmez O, Koç Ç K. Trace-driven cache attacks on AES(short paper)[M] // Information and Communications Security. Springer Berlin Heidelberg, 2006: 112-121

[27] Gallais J F, Kizhvatov I, Tunstall M. Improved trace-driven cache-collision attacks against embedded AES implementations [M] // Information Security Applications. Springer Berlin Heidelberg, 2011: 243-257

[28] Osvik D A, Shamir A, Tromer E. Cache attacks and countermeasures: the case of AES[M] // Topics in Cryptology-CT-RSA 2006. Springer Berlin Heidelberg, 2006: 1-20

[29] Tromer E, Osvik D A, Shamir A. Efficient cache attacks on AES, and countermeasures[J]. Journal of Cryptology, 2010, 23(1): 37-71

[30] 赵新杰，王韬，郭世泽，等. AES 访问驱动 Cache 计时攻击[J]. 软件学报, 2011, 22(3): 572-591