第 35 卷 第 5 期 计 算 机 工 程 2009 年 3 月
Vol.35 No.5 Computer Engineering March 2009
·网络与通信· 文章编号：1000—3428(2009)05—0108—03 文献标识码：A 中图分类号：TP301.6

# 基于 Bloom Filter 的报文分类算法

白建东，孙志刚
(国防科技大学计算机学院，长沙 410073)

**摘 要：**针对传统报文分类算法在实际运行中存在的问题，提出一种基于 Bloom Filter 的报文分类算法。将该算法的思想应用于入侵防护系统硬件模型，建立相应的流信息预处理引擎，并介绍具体的实现方法。实验结果表明，该算法是有效实用的。
**关键词：**Bloom Filter 技术；报文分类；入侵防护系统

# Packet Classification Algorithm Based on Bloom Filter

BAI Jian-dong, SUN Zhi-gang
(School of Computer, National University of Defense Technology, Changsha 410073)

**【Abstract】**Aiming at the problems exixted in running process of traditional packet classification algorithm, a novel packet classification algorithm based on Bloom Filter is proposed, and the idea of which is used into the hardware medel of Intrusion Prevention System(IPS). The corresponding pretreatment engine of stream information is set up, and the method of implementation is introduced. Experimental results show this algorithm is effective and has the value of application.
**【Key words】**Bloom Filter technology; packet classification; Intrusion Prevention System(IPS)

## 1 概述

随着互联网技术的发展，高效实用的快速报文分类算法成为近年来的研究热点，算法的目的是在有限空间及时间范围内，通过大规模的分类规则对报文进行快速分类处理。

传统的报文分类算法分为 4 类[1]：基于数据结构的算法，几何算法，启发式算法和基于硬件的算法。从实现的角度可将算法分为 2 种：全部采用硬件实现的算法以及软件实现的算法。软件算法一般提供特定维数的分类能力，特别是一维和二维的分类算法，但支持高维分类的算法在空间和时间复杂度上不能较好地满足要求。全部用硬件实现的算法对报文分类而言，分类速度快，时间和空间复杂度等均能满足要求，但价格较昂贵[2]。

## 2 Bloom Filter 简介

Bloom Filter 是种简单的、空间利用率高的、支持大容量数据集合查询的数据结构，其处理过程包括数据元素的插入、查询和删除。该数据结构主要包括 2 个部分：多个 Hash 函数和固定大小的特征向量，其结构如图 1 所示。

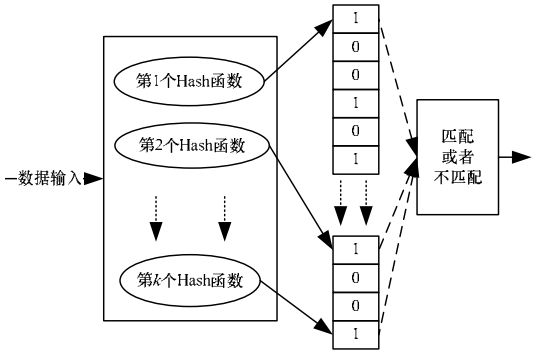


**图 1 Bloom Filter 结构**

为便于其工作原理的描述，在此假设 Bloom Filter 中 Hash 函数的数量为 $k$，特征向量深度为 $m$，可同时支持 $n$ 个数据元素的快速查询。

在插入阶段，对于拥有 $n$ 个数据元素的集合，需要对其中的每个元素进行 $k$ 个不同 Hash 函数的并行计算，得到 $j(j \leq k)$个大小介于 0~$m$-1 之间的 Hash key，以 Hash key 为地址访问特征向量，对应于其中的 $j$ 个比特位。若该位为 0，则将其置 1；若该位已被置 1，则保持不变。

对于查找操作，与数据插入的过程基本相同。首先将需要查询的数据经过 $k$ 个不同 Hash 函数的计算，以得到的 Hash key 为地址访问特征向量，如果其中的 $j$ 个对应位的值均为 1，则表示被查询的数据可能存在于数据集合中；如果至少有一个值为 0，则输入的数据肯定不在数据集合中。

删除操作比较简单，只需将特征向量中 $j$ 个对应位的数值由 1 变为 0 即可。

Bloom Filter 中多个 Hash 函数并行计算的机制在一定程度上有效解决了大容量数据集合的表示和查询问题，但在实际应用中可以发现多个 Hash 函数的冲突问题是无法避免和消除的。因此，Bloom Filter 在支持快速查询的同时，产生了假阳性。

假阳性是指被查询的数据实际上不在给定的数据集合中，但是该数据经过 Hash 计算得到的 Hash key 对应的特征向量中的值均为 1，从而产生错误的查询结果。根据分析可得假阳性发生的概率 $f$ 满足以下计算公式[3]：通过公式计算可知当 $k=(m/n)\ln 2$ 时，$f$ 取得最小值。

**基金项目：**国家“973”计划基金资助项目(2003CB314802)
**作者简介：**白建东(1983－)，男，硕士研究生，主研方向：计算机系统结构；孙志刚，副研究员
**收稿日期：**2008-09-05 **E-mail：**hnbjd2001@163.com

$$f=\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k}\approx\left(1-\mathrm{e}^{\frac{-kn}{m}}\right)^{k}$$

在对 Bloom Filter 的介绍中没有考虑到数据元素的删除问题：如果删除一个元素，那么它在特征向量中对应的 $j$ 个值由 1 变为 0，此时有可能影响集合中其他元素的查询，原因在于多个数据元素得到的 Hash key 存在部分或者全部相同的可能性。为解决该问题，有关研究人员提出计数型 Bloom Filter[3]，即特征向量中的每一位添加一个计数器，当 Bloom Filter 增加或删除一个集合元素时，同时增加或减少与该元素对应的计数器的值，此时需要考虑的问题是对计数器的大小要选取适当，避免出现溢出的情况。文献[3]指出：4 bit 宽度的计数器适用于大多数应用环境。

## 3 BFPC 算法描述及性能分析

BFPC 算法以报文分类快速、准确为原则，其基本思想是将计数型 Bloom Filter 与动态链表进行有效结合，实现报文分类的功能。图 2 为 BFPC 算法的原理结构。

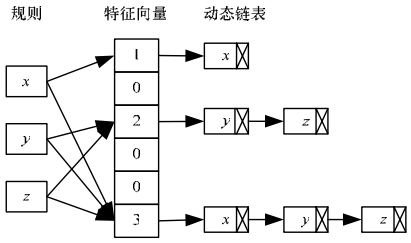


**图 2 BFPC 算法结构**

从图 2 可以看出，该算法具有 2 个特点：(1)运用计数型 Bloom Filter 代替标准的 Bloom Filter。计数型 Bloom Filter 不仅完全具备标准 Bloom Filter 的一切性能和优点，而且较好地解决了集合中规则删除时可能导致的查询失败问题；(2)在计数型 Bloom Filter 查询结果的基础上，引入动态链表执行规则的精确匹配。不论是标准 Bloom Filter 还是计数型 Bloom Filter，其假阳性的产生是不可避免的，只能通过对某些参数的设置来使其产生的概率最小。在此动态链表提供精确匹配的目的是及时对假阳性进行检测，对错误的匹配结果做出更正，提高 BFPC 算法报文分类结果的准确性。

BFPC 算法同其他算法一样，支持规则的添加、查询和删除请求。

规则查询算法描述如下：

(1)对需要查询的规则进行 Hash 并行计算，得到 $j(j\le k)$个 Hash key。

(2)以 Hash key 为地址并行访问特征向量，检查计数器数值，如果其中至少存在 1 个计数器为 0 的情况，表示查询失败，则转(1)，否则转(3)。

(3)根据特征向量存放的动态链表首地址执行并行访存，与链表中存放的规则进行精确匹配，如果第 1 个表项匹配失败，则转(4)，否则转(5)。

(4)根据表项中存放的后指针访问下一个表项，继续相同的匹配操作，直到动态链表的最后表项，若其中某个项匹配成功，则停止后续的匹配操作，返回匹配结果，转(6)。

(5)停止后续的匹配操作，返回匹配结果，转(6)。

(6)根据收到的多个匹配结果得出规则查询成功与否的结论，转(1)。

添加规则到动态链表的算法如下：

(1)将需要添加的规则先进行 Hash 计算，以得到的 $j(j\le k)$个 Hash key 为地址访问特征向量，如果某个计数器数值为 0，转(2)，否则转(3)。

(2)申请空闲的表项，将其地址存放到对应的特征向量中，同时保存规则到表项中，更改表项中的前指针为对应的特征向量地址，后指针为空，计数型数值加 1，转(4)。

(3)依据特征向量中存放的链表首地址依次访问找到最后的表项，申请空闲表项，将最后表项的后指针更改为空闲表项地址，保存规则到空闲表项中，同时将空闲表项的前指针更改为最后表项的地址，其后指针为空。计数型数值加 1，转(4)。

(4)若在 $j$ 个链表中规则均添加完成，转(1)。否则等待。

从动态链表中删除规则的算法如下：

(1)将需要删除的规则经过 Hash 计算，以得到的 $j(j\le k)$个 Hash key 为地址访问特征向量，并根据存放的链表头指针访问不同的链表。根据被删除规则在链表的不同位置执行不同的操作。

(2)若规则位于链表中第 1 个的位置，转(3)；若位于链表中间的位置，转(4)；若位于链表的最后的位置，转(5)；若为链表中的唯一存放的规则，转(6)。

(3)更改特征向量中的头指针为链表中第 2 表项的地址，将第 2 表项中的前指针改为特征向量的地址，转(7)。

(4)找到被删除表项的前后表项，分别更改前后表项的后、前指针为对方的地址，转(7)。

(5)找到被删除表项的前一项，更改其后指针为空，转(7)。

(6)更改特征向量中的链表头指针为空，转(7)。

(7)删除规则，回收存储空间，将计数器数值减 1，此时如果 $j$ 个删除操作都已经完成，则返回(1)，否则等待，直到所有删除操作完成为止。

下面将从时间复杂度、空间复杂度和更新复杂度 3 个方面对 BFPC 算法进行性能分析。为便于与其他算法比较，在此重新定义 BFPC 算法所支持查询的规则个数为 $n$，Hash 函数个数为 $k$，特征向量深度为 $m$。在理想情况下，Hash 计算和特征向量的访问均可在单个时钟周期内完成。针对 $n$ 个元素，共需要向特征向量产生 $nk$ 次映射。由于在 Hash key 均匀分布的前提下，每个计数器数值为 $nk/m$，因此 BFPC 算法的时间复杂度为 $O(2+nk/m)$。

BFPC 算法的空间应用主要包括特征向量和动态链表 2 个方面，特征向量的空间大小与其深度 $m$ 有关，动态链表的大小与 $n,k$ 有关，其空间复杂度为 $O(nk+m)$。基于上述的分析可知 BFPC 算法的更新复杂度为 $O(2+nk/m)$。当 $k$ 为 $(m/n)\ln 2$ 时，BFPC 算法的时间、空间和更新复杂度分别更新为 $O(2+\ln 2)$, $O(m(1+\ln 2))$和 $O(2+\ln 2)$。通过与其他算法的性能比较可以发现，BFPC 算法在性能上具有一定优势。

## 4 BFPC 算法思想在 IPS 中的具体应用

### 4.1 IPS 硬件模型及处理流程

本节将对在 IPS 中如何运用 BFPC 算法进行报文分类进行详细的研究，并建立相关的硬件模型。在此假设到达 IPS 的为经过分片重组后的报文，且按序到达。

由于网络中 90%以上的流量为 TCP 流量，因此在该模型中主要针对 TCP 协议的报文进行相关的预处理操作，对于非 TCP 协议的情况则可以直接进行过滤操作。具体的 IPS 硬件模型以及处理流程分别如图 3、图 4 所示。

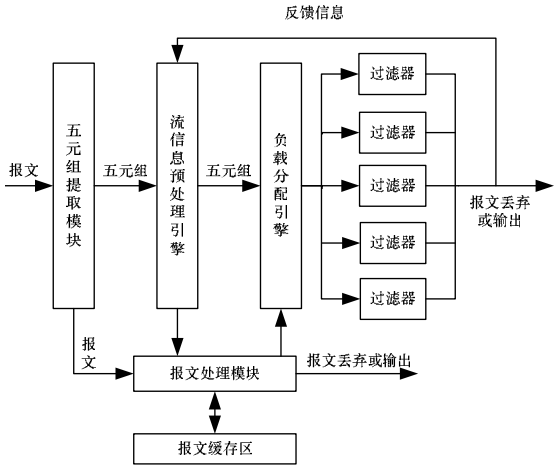


**图 3　IPS 硬件模型**

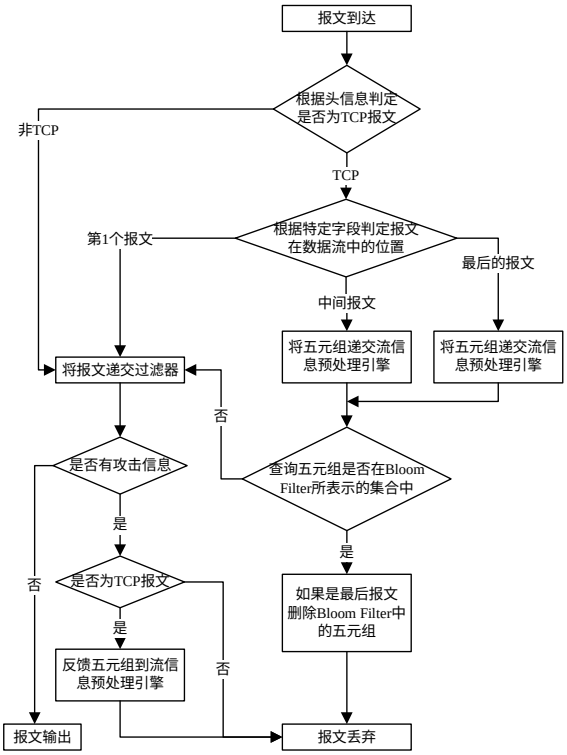


**图 4　IPS 硬件模型处理流程**

### 4.2　流信息预处理引擎实现方案

该引擎的作用在于对输入的五元组规则进行查询，判断其是否存在于特定的五元组集合中。该集合中存放的五元组规则代表的是含有恶意攻击信息的 TCP 数据流，如果查询匹配成功，则表示到达的报文不安全，否则不能断定报文安全与否。为便于该引擎的具体实现的说明，先设置参数的大小，如表 1 所示。

**表 1　主要参数**

| 参数 | 参数描述 | 数值 |
|---|---|---|
| *m* | 特征向量深度 | 16 Kb |
| *w* | 特征向量宽度 | 18 bit |
| *n* | 五元组规则个数 | 2 Kb |
| *k* | Hash 函数个数 | 5 |
| *f* | 假阳性产生概率 | 0.021 7 |

在本引擎的实现中涉及到的数据结构主要包括：特征向量，动态链表，空闲流 ID 表和定时器等，其相互关系如图 5 所示。

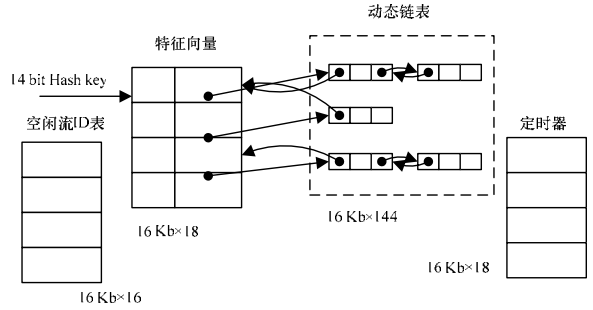


**图 5　主要数据结构关系**

为保证同时支持 $2\times10^3$ 条攻击数据流的检测，在此将特征向量设置成 16 Kb×18 的存储空间。其中，4 bit 用于存放计数器，另外的 14 bit 用于存放流 ID。

空闲流 ID 的作用是作为动态链表表项的地址使用。根据同时支持 $2\times10^3$ 个不同五元组记录以及 Hash 函数的个数为 5 的实际环境，空闲流 ID 表需要设置为 16 Kb×16 的表，其中的 14 bit 作为流 ID 的宽度，剩余的作为保留位。

动态链表保存五元组规则等节点信息，共有 16 Kb 个表项，每个 144 bit，分为 8 个 18 bit 进行存放，具体存放内容如表 2 所示。

**表 2　动态链表表项存储信息**

| 偏移 | 内容(18 bit) |
|---|---|
| 0 | 40 000, head_pointer[13:0] |
| 1 | 40 000, tail_pointer[13:0] |
| 2 | P_type, 36 000, Stream_ID[13:0] |
| 3 | Stream_ID[31:14] |
| 4 | Stream_ID[49:32] |
| 5 | Stream_ID[67:50] |
| 6 | Stream_ID[85:68] |
| 7 | Stream_ID[103:86] |

在表 2 中，head_pointer[13:0]表示 14 bit 的前项指针，tail_pointer[13:0]表示 14 bit 的后项指针。P_type 为标示位，为 1 表示前项指针指向特征向量的地址，即本表项为链表的第 1 个表项，为 0 表示前项指针指向的是链表的前一个表项。Stream_ID[103:0]表示 104 bit 的五元组规则。

定时器设置的目的是解决数据流超时问题。分别为每个五元组规则对应一个定时器。在此设置该表为 16 Kb×18 的大小，其中，1 bit 用于表示流的状态，比如 1 表示数据流存在，0 表示不存在。其余的宽度可以用来存放定时器。假设数据流超时时间为 16 s，在此将表项的[3:0]设置为定时器，余下的宽度保留。当该流有报文到达时，将定时器清 0，定时器每隔 1 s 加 1，当定时器计满为 0xf 时，表示该流已经有 16 s 没有报文到达，可认为该流超时。

该引擎的处理流程分为 3 种情况：首先考虑查询请求的情形。对于五元组提取模块发送的查询请求，Bloom Filter 先对输入的五元组执行 5 个 Hash 函数的计算，根据计算得到的 Hash key 访问特征向量。根据存放的 14 bit 的流 ID 作为首地址访问片外存储器中的动态链表，执行五元组规则的精确匹配。根据匹配的结果对报文做出不同的处理。如果匹配成功，除了丢弃报文以外，还需要向超时流管理发送更新定时器的信息，将定时器及时清 0。

如果某个五元组规则需要添加到该引擎的动态链表中，首先仍将该五元组作为 Hash 计算的对象，通过 5 个 Hash 函

（下转第 124 页）