



## (12) 发明专利

(10) 授权公告号 CN 102195866 B

(45) 授权公告日 2014. 05. 14

(21) 申请号 201010521059. 7

H04L 29/06 (2006. 01)

(22) 申请日 2010. 10. 27

(56) 对比文件

(30) 优先权数据

10-2010-0019209 2010. 03. 03 KR

10-2010-0081704 2010. 08. 24 KR

CN 101577721 A, 2009. 11. 11, 全文.

CN 101655861 A, 2010. 02. 24, 全文.

CN 1881950 A, 2006. 12. 20, 全文.

US 7436830 B2, 2008. 10. 14, 全文.

(73) 专利权人 梨花女子大学产学协力团

地址 韩国首尔市西大门区大岬洞梨花女子  
大学

审查员 王健

(72) 发明人 林惠淑

(74) 专利代理机构 上海翼胜专利商标事务所

(普通合伙) 31218

代理人 翟羽

(51) Int. Cl.

H04L 12/745 (2013. 01)

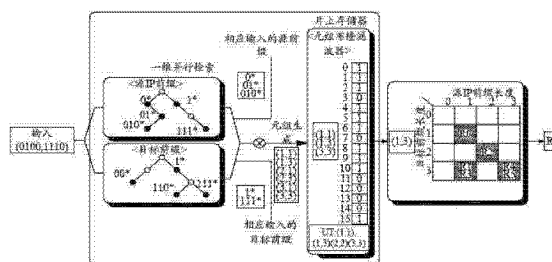
权利要求书3页 说明书14页 附图4页

### (54) 发明名称

利用布隆滤波器的数据包分类方法及数据包  
分类装置

### (57) 摘要

本发明涉及一种利用了布隆滤波器的数据包分类方法及数据包分类装置,包括:(a) 对输入数据包的一个以上的字段进行逐字段的检索,从而判断每个所述一个以上的字段的字段值与存储于规则集的相应的字段的字段值相互一致的长度(以下,称为“一致长度”),并且生成出由相应于每个所述一个以上的字段的所述一致长度的组合(Combination)所构成的元组(Tuple)列表;(b) 从所述元组列表中只将出存在于所述规则集的元组挑选出来;(c) 利用布隆滤波器(BloomFilter)过滤每个而挑选出来的元组;及(d) 只将过滤后的结果为阳性(Positive)的元组作为检索对象,并检索最佳匹配规则(BMR:BestMatchingRule)。本发明除了利用字段长度的数据,还利用字段值的数据,从而减少检索对象元组数,因此可以减少存储器存取次数,还可以提高检索性能。



1. 一种利用了布隆滤波器的数据包分类方法,包括:

(a) 对输入数据包的一个以上的字段进行逐字段的检索,从而判断每个所述一个以上的字段的字段值与存储于规则集的相应的字段的字段值相互一致的长度,以下将“每个所述一个以上的字段的字段值与存储于规则集的相应的字段的字段值相互一致的长度”称为“一致长度”,并且生成出由相应于每个所述一个以上的字段的所述一致长度的组合(Combination)所构成的元组(Tuple)列表;

(b) 从所述元组列表中挑选出存在于所述规则集的特定元组;

(c) 对每个挑选出来的元组利用布隆滤波器进行过滤;及

(d) 只将过滤后的结果为阳性(Positive)的元组作为检索对象,并检索最佳匹配规则(BMR:Best Matching Rule),

其中所述步骤(c)包括:

(c1) 对每个所述被挑选出来的元组生成散列索引;

(c2) 存取所述散列索引所指的所述布隆滤波器的条目,并确认所述布隆滤波器的条目的比特值;及

(c3) 只挑选出比特值为阳性的元组来作为检索对象。

2. 如权利要求1所述的利用了布隆滤波器的数据包分类方法,其特征在于,

存取所述散列索引所指的所述布隆滤波器的条目后,对存储于散列表的条目的规则进行检索,并从被检索的规则中决定出所述最佳匹配规则。

3. 如权利要求1所述的利用了布隆滤波器的数据包分类方法,其特征在于,

于所述步骤(c1),分别从所述一个以上的字段中取出所述被挑选出来的元组所表示的所述一致长度大小的较高序位比特后,相互串接(Concatenation)起来生成字符串(String),并由所生成的字符串(String)生成所述散列索引。

4. 如权利要求1所述的利用了布隆滤波器的数据包分类方法,其特征在于,

于所述步骤(c1),从各个所述一个以上的字段中取出所述挑选出来的元组所表示的所述一致长度大小的较高序位比特,再对取出的各值附加一定的位模式以固定长度值进行填补(Padding)后,由相互串接被填补的值而生成的字符串(String)生成所述散列索引。

5. 如权利要求3或4所述的利用了布隆滤波器的数据包分类方法,其特征在于,

所述步骤(c1)包括:

将所述字符串输入至CRC生成器(Cyclic Redundancy Check Generator)而生成固定长度字符串;及

由所述固定长度字符串生成多个所述散列索引。

6. 如权利要求1所述的利用了布隆滤波器的数据包分类方法,其特征在于,

所述一个以上的字段为源前缀(Source Prefix)字段与目标前缀(Destination Prefix)字段。

7. 如权利要求1所述的利用了布隆滤波器的数据包分类方法,其特征在于,

所述步骤(a)中,对所述输入数据包的所述一个以上的字段进行逐字段的检索为利用了二分法检索级别(Binary Search on Level)的一维检索。

8. 如权利要求1所述的利用了布隆滤波器的数据包分类方法,其特征在于,所述步骤(d)包括:

(d1) 对于属于所述检索对象的元组生成散列表用散列索引；

(d2) 存取所述散列表用散列索引所指的散列表条目,并对所述散列表的条目的规则进行检索;及

(d3) 从检索出来的规则中决定出所述最佳匹配规则。

9. 一种利用了布隆滤波器的数据包分类装置,其特征在于,包括:

存储器,用于存储具备了用于数据包分类的规则的散列表;

元组生成部,对输入数据包的一个以上的字段进行逐字段的检索,从而判断每个所述一个以上的字段的字段值与存储于规则集的相应的字段的字段值相互一致的长度,以下将“每个所述一个以上的字段的字段值与存储于规则集的相应的字段的字段值相互一致的长度”称为“一致长度”,生成出由相应于每个所述一个以上的字段的所述一致长度的组合所构成的元组(Tuple)列表,并且从所述元组列表中挑选出存在于所述规则集中的特定元组;及

数据包分类控制部,利用布隆滤波器对每个由所述元组生成部而挑选出来的元组执行过滤,只将过滤后结果为阳性(Positive)的元组作为检索对象,并存取所述散列表而检索最佳匹配规则(BMR:Best Matching Rule),所述数据包分类控制部包括:

散列部,对每个所述挑选出来的元组生成散列索引;

所述布隆滤波器,确认所述散列索引所指的条目的比特值后,只将所述比特值为阳性的元组作为检索对象而进行挑选;及

规则检索部,利用对应于属于所述检索对象的元组的所述散列索引,从而存取所述散列表的条目后检索存储于所述散列表的条目的规则,并从检索到的规则中决定出最佳匹配规则。

10. 如权利要求9所述的利用了布隆滤波器的数据包分类装置,其特征在于,所述散列部包括:

CRC生成器(Cyclic Redundancy Check Generator),接收一字符串(String)后生成固定长度字符串,所述字符串是从各个所述一个以上的字段取出所述挑选出来的元组所表示的所述一致长度大小的较高序位比特后相互串接(Concatenation)生成;及

散列索引生成部,由所述固定长度字符串生成多个所述散列索引。

11. 如权利要求9所述的利用了布隆滤波器的数据包分类装置,其特征在于,所述散列部包括:

CRC生成器(Cyclic Redundancy Check Generator),接收一字符串(String)后生成固定长度字符串,所述字符串是从各个所述一个以上的字段取出所述挑选出来的元组所表示的所述一致长度大小的较高序位比特,再对每个取出的值附加一定的位模式而填补(Padding)至固定长度后,互相串接被填补的值而生成;及

散列索引生成部,由所述固定长度字符串生成多个所述散列索引。

12. 如权利要求9所述的利用了布隆滤波器的数据包分类装置,其特征在于,所述一个以上的字段为源前缀字段与目标前缀字段。

13. 如权利要求9所述的利用了布隆滤波器的数据包分类装置,其特征在于,所述一个以上的字段为源端口号字段与目标端口号字段。

14. 如权利要求10所述的利用了布隆滤波器的数据包分类装置,其特征在于,

所述元组生成部中所执行的所述逐字段的检索为利用了二分法检索级别的一维检索。

15. 一种利用了布隆滤波器的数据包分类装置,其特征在于,包括:

存储器,用于存储具备了用于数据包分类的规则散列表;

元组生成部,对输入数据包的一个以上的字段进行逐字段的检索,从而判断每个所述一个以上的字段的字段值与存储于规则集的相应的字段的字段值相互一致的长度,以下将“每个所述一个以上的字段的字段值与存储于规则集的相应的字段的字段值相互一致的长度”称为“一致长度”,生成出由相应于每个所述一个以上的字段的所述一致长度的组合所构成的元组(Tuple)列表,并且从所述元组列表中挑选出存在于所述规则集中的特定元组;及

数据包分类控制部,利用布隆滤波器对每个由所述元组生成部而挑选出来的元组执行过滤,只将过滤后结果为阳性(Positive)的元组作为检索对象,并存取所述散列表而检索最佳匹配规则(BMR:Best Matching Rule),所述数据包分类控制部包括:

散列部,对于每个所述挑选出来的元组生成布隆滤波器用散列索引与散列表用散列索引;

所述布隆滤波器,确认所述布隆滤波器用散列索引所指的条目的比特值后,只将所述比特值为阳性的元组作为检索对象而进行挑选;及

规则检索部,利用属于所述检索对象的元组所对应的所述散列表用散列索引,存取所述散列表的条目后检索存储于所述散列表的条目的规则,并从检索到的规则中决定出所述最佳匹配规则。

16. 如权利要求15所述的利用了布隆滤波器的数据包分类装置,其特征在于,所述散列部包括:

CRC生成器(Cyclic Redundancy Check Generator),接受一字符串(String)后生成固定长度字符串,所述字符串是从各个所述一个以上的字段取出所述挑选出来的元组所表示的所述一致长度大小的较高序位比特,并相互串接(Concatenation)起来而生成;及

散列索引生成部,由所述固定长度字符串生成多个所述布隆滤波器用散列索引及一个所述散列表用散列索引。

17. 如权利要求15所述的利用了布隆滤波器的数据包分类装置,其特征在于,所述散列部包括:

CRC生成器(Cyclic Redundancy Check Generator),接收一字符串(String)后生成固定长度字符串,所述字符串是从各个所述一个以上的字段取出所述挑选出来的元组所表示的所述一致长度大小的较高序位比特,对每个取出的值附加一定的位模式而填补(Padding)至固定长度后,互相串接被填补的值而生成;及

散列索引生成部,由所述固定长度字符串生成多个所述布隆滤波器用散列索引及一个所述散列表用散列索引。

## 利用布隆滤波器的数据包分类方法及数据包分类装置

### 技术领域

[0001] 本发明涉及一种利用布隆滤波器 (Bloom Filter) 的数据包分类方法及数据包分类装置。更具体而言,涉及一种数据包分类方法及数据包分类装置,利用布隆滤波器对根据输入数据包的字段长度一致数据而生成的元组列表进行过滤,从而只对有可能与规则集一致的元组实施检索。

### 背景技术

[0002] 在该部分进行叙述的内容仅单纯地用于提供本发明实施例的背景数据,并不能构成现有技术。

[0003] 为了提供应用程序所需要的各种服务,互联网路由器需要利用输入数据包的标头数据来决定数据包 (Packet) 的类别 (Class),并提供各类别所相应的适当的服务。决定出数据包所属类别的操作称为数据包分类,数据包分类主要使用源 (Source) IP 地址、目标 (Destination) IP 地址、源端口号、目标端口号、协议字段等来执行。

[0004] 数据包分类具有如下困难。

[0005] 第一,数据包分类中多维检索必不可少,且各字段别的检索方法不同。IP地址字段需执行最长长度一致方式,端口号字段需执行域一致方式,还有协议字段需确定一致。由于需要同时执行各种不同方式的检索,因此检索复杂且检索速度降低。

[0006] 第二,需要附加一过程,通过检索找出所有一致的规则,并从中选择出最优先规则。

[0007] 第三,所有这些操作需要相应于被输入的数据包的速度,并以线速进行处理。

[0008] 因此为了提高路由器的数据包处理速度而进行着很多研究。特别是为了克服多维检索的困难而提出了仅使用一部分字段对数据包进行分类的方法,该方法相比于使用所有字段来对数据包进行分类的方法可以改善检索速度,且具有代表性的有元组 (Tuple) 空间检索算法。

[0009] 【表 1】。

[0010]

| RuleNo. | Source Prefix | Destination Prefix | Tuple    |
|---------|---------------|--------------------|----------|
| R0      | 1 *           | 1 *                | ( 1, 1 ) |
| R 1     | 1 *           | 1 1 0 9            | (1, 3)   |
| R 2     | 0 1 *         | 0 0 *              | (2, 2 )  |
| R3      | 0 *           | 1 1 1 *            | (1, 3)   |
| R4      | 111 *         | 110 *              | (3, 3)   |

|    |       |         |        |
|----|-------|---------|--------|
| R5 | 010 * | 1 1 0 * | (3, 3) |
|----|-------|---------|--------|

[0011] 所谓的元组空间为由各字段的长度组合构成的空间,如表 1 中举例说明的规则集,其中 R0 的源前缀(prefix)的长度为 1,且目标前缀的长度为 1,故属于 (1, 1) 元组, R1 的源前缀的长度为 1,且目标前缀的长度为 3,故属于 (1, 3) 元组。

[0012] 如表 1 所示,如果定义的是由源前缀与目标前缀的长度组合构成的元组空间时,前缀可以具有 0 至 32 之间的长度,因此最多可以生成  $33 \times 33 = 1089$  个的元组。因此,当对输入于路由器的各输入数据包中哪怕存在一个规则的所有元组进行线性检索时,需要消耗很多时间,为了减少需要检索的元组数量而进行很多的研究,其中具有代表性的有元组剪枝算法(Tuple Pruning Algorithm)。

[0013] 元组剪枝算法使用各个字段类别的检索结果而消除检索中所不必要的元组,从而改善检索性能。对源前缀字段与目标前缀字段分别进行一维检索,将由该结果得出的一致的前缀长度数据进行组合,由此形成元组。组合两个字段的长度数据需要交叉组合(Cross-Producting)过程,而该过程是对于各字段的检索结果中与输入一致的前缀的长度生成出所有具有可能性的元组数的过程。但是,由交叉组合的结果而所得到的元组由于是以各个字段类别而一致的前缀长度的组合,因此有可能会生成出规则集(rule set)中并不实际存在的元组。因此除去规则集中不存在的元组,决定出最终进行检索的元组。

[0014] 图 1 为现有的元组剪枝算法的示意图。

[0015] 对表 1 的规则集作出用于一维检索的特里结构(Trie),并在元组空间储存规则。举例说明在输入数据包的源前缀与目标前缀为 (0100, 1110) 的情况下进行检索的过程。

[0016] 首先,对源前缀字段与目标前缀字段同时进行各自的检索,并找出所有一致的长度。源前缀字段的检索结果由于与  $0*$ 、 $01*$ 、 $010*$  一致,因此判断出长度 1、2、3 为一致长度,而在目标前缀字段则  $1*$  与  $111*$  一致,因此判断出长度 1、3 为一致长度。对一维检索结果中一致的前缀的长度数据执行交叉组合时总共形成 (1, 1)、(1, 3)、(2, 1)、(2, 3)、(3, 1)、(3, 3) 6 个元组。这时通过与预存的元组列表的比较,并从已形成的元组中除去规则集中并不存在的元组 (2, 1)、(2, 3)、(3, 1)。从而最终剩下的元组总共有 (1, 1)、(1, 3)、(3, 3) 这 3 个。

[0017] 对于该三个元组进行的输入 (0100, 1110) 的检索如下。首先,将连接了第一个字段的第一个比特与第二个字段的第一个比特的字符串 '01' 作为散列键(Hash key)来对 (1, 1) 元组执行使用了散列法的检索。接下来以相同方法对 (1, 3) 元组进行 '0111' 的检索,最后对 (3, 3) 元组进行 '010111' 的检索,从而找出最佳匹配规则(BMR:Best Matching Rule)—R3。这样使用按字段类别对给定的输入进行检索而除去不需要的元组的方式,从而可以提高检索速度。

[0018] 但是在元组剪枝算法中,元组是由前缀的长度组合所决定的,因此无法滤出前缀的长度组合存在但前缀的值却相异的情况。前面举出的示例中,被过滤后剩余的元组 (1, 1)、(1, 3)、(3, 3) 中的 (1, 1) 与 (3, 3) 元组中所存储的规则与给定的输入明显不一致,但是并没有被除去,而是被留了下来,由此导致了不需要的元组检索。

[0019] 即,元组剪枝算法由于只根据前缀的长度数据来除去元组,因此会导致不必要的元组检索,因此存在降低检索性能的缺点。

## 发明内容

[0020] 为了解决上述问题,本发明的目的在于提供一种利用布隆过滤器对根据输入数据包的字段长度数据而生成的元组列表进行过滤,从而只对有可能与规则集一致的元组进行检索的数据包分类方法及数据包分类装置。

[0021] 为了达到上述目的,本发明提供一种数据包分类方法,包括:(a) 对输入数据包的一个以上的字段进行逐字段的检索,从而判断每个所述一个以上的字段的字段值与存储于规则集的相应的字段的字段值相互一致的长度,以下将“每个所述一个以上的字段的字段值与存储于规则集的相应的字段的字段值相互一致的长度”称为“一致长度”,并且生成由相应于每个所述一个以上的字段的所述一致长度的组合(Combination)所构成的元组(Tuple)列表;(b) 从所述元组列表中只将出存在于所述规则集的元组挑选出来;(c) 利用布隆过滤器(BloomFilter)对每个而挑选出来的元组进行过滤;及(d) 只将过滤后的结果为阳性(Positive)的元组作为检索对象,并检索最佳匹配规则(BMR:Best Matching Rule)。所述步骤(c)包括:(c1) 对每个所述被挑选出来的元组生成散列索引;(c2) 存取所述散列索引所指的所述布隆过滤器的条目,并确认所述布隆过滤器的条目的比特值;及(c3) 只挑选出比特值为阳性的元组来作为检索对象。

[0022] 而且,本发明还提供数据包分类装置,包括:元组生成部,对存储器输入数据包的一个以上的字段进行逐字段的检索,其中,所述存储器输入数据包存储具备有用于数据包分类的规则散列表,从而判断每个所述一个以上的字段的字段值与存储于规则集的相应的字段的字段值相互一致的长度,以下将“每个所述一个以上的字段的字段值与存储于规则集的相应的字段的字段值相互一致的长度”称为“一致长度”,生成由相应于每个所述一个以上的字段的所述一致长度的组合所构成的元组(Tuple)列表,并且从所述元组列表中只挑选出所述规则集中存在的元组;及数据包分类控制部,利用布隆过滤器对每个由所述元组生成部而挑选出来的元组执行过滤,并且只将过滤后结果为阳性(Positive)的元组作为检索对象,并存取所述散列表而检索最佳匹配规则(BMR:Best Matching Rule)。所述数据包分类控制部包括:散列部,对每个所述挑选出来的元组生成散列索引;所述布隆过滤器,确认所述散列索引所指的条目的比特值后,只将所述比特值为阳性的元组作为检索对象而进行挑选;及规则检索部,利用对应于属于所述检索对象的元组的所述散列索引,从而存取所述散列表的条目后检索存储于所述散列表的条目的规则,并从检索到的规则中决定出最佳匹配规则。本发明还提供数据包分类装置,包括:存储器,用于存储具备了用于数据包分类的规则散列表;元组生成部,对输入数据包的一个以上的字段进行逐字段的检索,从而判断每个所述一个以上的字段的字段值与存储于规则集的相应的字段的字段值相互一致的长度,以下将“每个所述一个以上的字段的字段值与存储于规则集的相应的字段的字段值相互一致的长度”称为“一致长度”,生成由相应于每个所述一个以上的字段的所述一致长度的组合所构成的元组(Tuple)列表,并且从所述元组列表中挑选出存在于所述规则集中的特定元组;及数据包分类控制部,利用布隆过滤器对每个由所述元组生成部而挑选出来的元组执行过滤,只将过滤后结果为阳性(Positive)的元组作为检索对象,并存取所述散列表而检索最佳匹配规则(BMR:Best Matching Rule)。所述数据包分类控制部包括:散列部,对于每个所述挑选出来的元组生成布隆过滤器用散列索引与散列表用散

列索引；所述布隆滤波器，确认所述布隆滤波器用散列索引所指的条目的比特值后，只将所述比特值为阳性的元组作为检索对象而进行挑选；及规则检索部，利用属于所述检索对象的元组所对应的所述散列表用散列索引，存取所述散列表的条目后检索存储于所述散列表的条目的规则，并从检索到的规则中决定出所述最佳匹配规则。

[0023] 根据本发明，不仅利用前缀的长度数据，还利用前缀值数据来减少检索对象元组的数量，从而减少存取存储器的次数，可以提高检索性能。

## 附图说明

[0024] 图 1 为现有的元组剪枝算法的示意图。

[0025] 图 2 为根据本发明实施例的利用了布隆滤波器的数据包分类方法的示意图。

[0026] 图 3 为根据本发明实施例的利用了布隆滤波器的数据包分类装置的示意图。

[0027] 图 4 为根据本发明第一实施例的数据包分类控制部的构成示意图。

[0028] 图 5 为根据本发明第二实施例的数据包分类控制部的构成示意图。

[0029] 图 6 为本发明实施例中所使用的 32 比特 CRC 生成器的结构示意图。

[0030] 图 7 及图 8 为现有的元组剪枝算法与根据本发明实施例的数据包分类方法在存储器存取次数方面的性能比较图形。

## 具体实施方式

[0031] 以下，通过举例的附图对本发明的部分实施例进行详细说明。需要注意的是，对各附图的构成要件添加参照符号时，即使相同的构成要件被标注在不同的附图上，也尽量使其具有同一符号。而且，在说明本发明时，如果判断相关的公知结构或功能的说明会影响到本发明揭示的清楚程度时，会省略其详细说明。

[0032] 而且，对构成本发明的构成要件进行说明时，会使用第一、第二、A、B、(a)、(b) 等的术语。这样的术语只是为了将其构成要件与其它构成要件区别开来，并不会被其术语而限制该当构成要件的本质、次序或顺序等。当记载某构成要件与其它构成要件进行“连接”、“接合”或“结合”时，该构成要件有可能直接连接或结合于该其它构成要件，但应该理解为还有各构成要件之间通过另一构成要件间接“连接”、“接合”或“结合”的可能性。

[0033] < 发明的构成 >

[0034] 根据本发明实施例的数据包分类方法包括：(a) 元组列表生成步骤；(b) 从元组列表中只挑选出存在于规则集的元组的步骤；(c) 利用布隆滤波器的过滤步骤；及 (d) 规则检索步骤。

[0035] 以下，参照图 2 对各步骤进行详细说明。

[0036] 图 2 为根据本发明实施例的利用了布隆滤波器的数据包分类方法的示意图。

[0037] 本发明的实施例中，举例说明了为了元组生成及规则检索而使用源前缀字段与目标前缀字段的情况。但是，这只不过是便于进行说明的实例，并不能用于限定本发明的思想。即，可以适当选择用于元组生成及规则的检索而使用的字段的种类和字段的数量，不管使用哪种字段或使用几个字段，可以说都属于本发明的技术思想。例如，当使用两个字段的的情况下，可以使用源端口号字段与目标端口号字段，当使用三个字段的的情况下，可以使用源前缀字段、目标前缀字段及源端口号字段等。而且，还可以使用四个或五个字段。

[0038] (a) 元组列表生成

[0039] 在元组列表生成步骤,对每个输入数据包的源前缀字段与目标前缀字段进行按逐字段的检索,从而生成逐字段的一致长度的组合构成的元组列表。即,对输入数据包的源前缀字段的字段值与存储于规则集的源前缀的值进行比较而判断出值相同的一致长度,并对输入数据包的目標前缀字段的字段值与规则集的目标前缀的值进行比较后判断出一致长度。

[0040] 例如,如图 2 所示,若输入数据包的源前缀与目标前缀为 (0100, 1110),而规则集如表 1 时,输入数据包的源前缀与表 1 的规则集的源前缀 0\*、01\*、010\* 一致,从而一致长度为 1、2、3。而且,输入数据包的目標前缀与规则集的目标前缀 1\*、111\* 一致,因此一致长度为 1、3。从而,由源前缀字段与目标前缀字段的一致长度的组合构成的元组有 (1, 1)、(1, 3)、(2, 1)、(2, 3)、(3, 1)、(3, 3),该些元组被生成为元组列表。

[0041] 其中,对各字段执行的逐个字段的检索可以通过二分法检索级别(Binary Search onLevel)的一维检索而执行。

[0042] (b) 元组挑选

[0043] 在元组挑选步骤,从属于元组列表的元组中挑选出存在于规则集的特定元组。

[0044] 例如,表 1 的规则集中只存在 (1, 1)、(1, 3)、(2, 2)、(3, 3) 的元组。因此,对于输入 (0100, 1110) 而生成的 (1, 1)、(1, 3)、(2, 1)、(2, 3)、(3, 1)、(3, 3) 总共 6 个元组中除去规则集中不存在的元组 (2, 1)、(2, 3)、(3, 1),如图 2 所示,只挑选出 (1, 1)、(1, 3)、(3, 3) 的元组。

[0045] (c) 利用布隆滤波器过滤

[0046] 利用布隆滤波器对在步骤 (b) 挑选出来的元组进行过滤,并只将过滤后的结果为阳性 (Positive) 的元组作为检索对象而进行挑选。

[0047] 布隆滤波器是用于确认输入是否为属于特定集合的要件的过滤器,将属于该特定集合的要件的比特值编程(programming)为 1 的情况下,当被输入某个输入值时通过询问过程而判断该输入值相对应的比特值是否为 1。当比特值为 1 时判断为该输入值有可能属于特定集合(该情况称为“阳性 (Positive)”),当比特值为 0 时,判断该输入值不属于特定集合(该情况称为“阴性 (Negative)”)。

[0048] 为了利用布隆滤波器而进行过滤,首先需要一将属于规则集的各规则对布隆滤波器进行编程的过程,以下参照表 2 对此进行说明。

[0049] 【表 2】

[0050]

| Rule No. | Source Prefix | Destination Prefix | Tuple  | Concatenated String | Hashing Code | BF Index (MSB, LSB by 4 bits) |
|----------|---------------|--------------------|--------|---------------------|--------------|-------------------------------|
| R0       | 1*            | 1*                 | (1, 1) | 11                  | 01101001     | 6, 9                          |
| R1       | 1*            | 110*               | (1, 3) | 1110                | 10000010     | 8, 2                          |
| R2       | 01*           | 00*                | (2, 2) | 0100                | 01101100     | 6, 10                         |
| R3       | 0*            | 111*               | (1, 3) | 0111                | 00000101     | 0, 5                          |
| R4       | 111*          | 110*               | (3, 3) | 111110              | 00010100     | 1, 4                          |
| R5       | 010*          | 110*               | (3, 3) | 010110              | 10101111     | 10, 15                        |

[0051] 表 2 为用于说明将表 1 的规则集对布隆过滤器进行编程的方法的表。

[0052] 首先串接 (Concatenation) 各规则的源前缀与目标前缀,从而生成一个字符串。例如,规则 R0 的情况下,由于源前缀为 1\*,目标前缀为 1\*,因此对于 R0 的字符串为 11。R1 的情况下,由于源前缀与目标前缀分别为 1\*、110\*,因此相互串接而生成 1110 的字符串。虽然在这里是串接两个前缀来使用,但并不限于此。

[0053] 这样对所有规则生成字符串,再将此作为散列函数的输入而使用,从而生成散列码。在本发明的实施例中使用了 CRC 生成器 (Cyclic Redundancy Check Generator) 作为散列函数。

[0054] CRC 生成器被输入各种长度的字符串后可以生成固定长度的散列码。因此可以从 CRC 生成器所生成的固定长度的散列码取出所需长度、所需数量的散列索引。为了说明 CRC 生成器的工作,举出图示了 32 比特 CRC 生成器结构的图 6 作为事例进行说明, CRC 生成器内的所有正反器 (Flip-flop) 被初始化为 0,输入值以一个比特为单位而进入时对存储于正反器的值与被输入的比特值进行 XOR 运算,直到最后的比特进入后 CRC 生成器的工作会停止,并存储于各正反器的值以 CRC 码输出,从而可以得到 32 比特的固定长度散列码。

[0055] 表 2 中的“Hashing Code (散列码)”项目为对各规则的两个前缀进行串接而生成的字符串输入至 8 比特 CRC 生成器而所得到的 CRC 码。

[0056] 得到散列码后,由此生成用于对布隆过滤器进行编程的散列索引。例如,对于 R0 来说,取出由 CRC 生成器得到的散列码“01101001”的最高有效 4 比特与最低有效 4 比特时,可以得到“0110”(6)与“1001”(9),从而将布隆滤波器的第 6 个条目与第 9 个条目的比特值编程为 1。

[0057] 表 2 的“BF index”表示通过如上述的方法生成的散列索引,通过该散列索引而被编程的布隆过滤器如图 2 所示。利用这样被编程的布隆过滤器而对步骤 (b) 中挑选出来的元组进行过滤。

[0058] 关于输入 (0100, 1110),举出对步骤 (b) 中挑选出来的元组 (1, 1)、(1, 3)、(3, 3) 的过滤过程作为实例来进行说明时,首先生成各元组相应的散列索引。例如,元组 (1, 1) 表示源前缀与目标前缀的一致长度均为 1,因此从输入 (0100, 1110) 的源前缀与目标前缀各取出较高序位 1 比特,并生成作为散列函数的输入而使用的字符串“01”。元组 (1, 3) 的情况下,则由于源前缀与目标前缀的一致长度分别为 1、3,因此从输入的源前缀取出较高序位 1 比特,从目标前缀取出较高序位 3 比特,从而生成字符串“0111”。对于元组 (3, 3) 同样执

行相同的过程,则会生成字符串“010111”。

[0059] 将各元组 (1, 1)、(1, 3) 及 (3, 3) 所相应的字符串“01”、“0111”及“010111”输入至 8 比特的 CRC 生成器时,会得到散列码“10110001”、“00000101”、“00011110”。

[0060] 由元组 (1, 1) 的字符串“01”所生成的散列码“10110001”中,分别各取出最高有效 4 比特与最低有效位 4 比特时,可以得到“1011”(11) 与“0001”(1) 的散列索引。从而使用该散列索引而接近布隆滤波器的第 1 个条目与第 11 个条目而确认各条目的比特值,由于第 1 个条目的比特值为 1,而第 11 个条目的比特值为 0,因此可以判断出元组 (1, 1) 不包括于规则集。因此,元组 (1, 1) 从检索对象中被除去。

[0061] 接着,由元组 (1, 3) 的字符串“0111”所生成的散列码“00000101”中,以如上所述的方式,分别取出最高有效 4 比特与最低有效 4 比特时,可以得到“0000”(0) 与“0101”(5) 的散列索引。利用该散列索引而接近布隆滤波器的第 0 个条目与第 5 个条目而确认比特值,由于第 0 个条目与第 5 个条目的比特值均为 1,因此元组 (1, 3) 所表示的 (0\*, 111\*) 有可能存在于规则集中。因此元组 (1, 3) 会成为检索对象元组。

[0062] 最后,以如上所述的方法对元组 (3, 3) 进行过滤时,散列码“00011110”的最高有效 4 比特散列索引所示意的第 1 个条目的比特值是 1,而最低有效 4 比特散列索引所示意的第 14 个条目的比特值为 0,因此可以判断为不包括在规则集,元组 (3, 3) 从检索对象中被除去。

[0063] (d) 规则检索

[0064] 规则检索步骤为只将布隆过滤器进行过滤的结果为阳性的元组作为检索对象,并决定出与输入数据包最为一致的最佳匹配规则 (BMR:Best Matching Rule)。

[0065] 参照表 2 进行更为详细的说明,首先,利用对布隆过滤器进行编程时使用的两个散列索引,散列表的条目存储有相应的规则。其中,为了对布隆过滤器进行编程而使用的两个散列索引可以都使用,但在本实施例中只使用了其中一个散列索引。例如,假设利用从散列码中取出最高有效 4 比特而所生成的散列索引而存储规则时,散列表的第 6 个条目中被存储规则 R0,第 8 个条目中被存储 R1,而规则 R2 ~ R5 均被存储于第 1 个散列索引所指的各条目。

[0066] 在之前步骤 (c) 中,只有元组 (1, 3) 被挑选为检索对象。从而,利用元组 (1, 3) 所对应的散列索引即“0000”(0) 来执行检索,这时存储于散列表的第 0 个条目的规则为 R3,而 R3 与输入数据包 (0\*, 111\*) 一致,因此决定 R3 作为最佳匹配规则。

[0067] 在以上说明的规则检索步骤 (d) 中,所使用的用于接近布隆滤波器的散列索引与用于接近散列表到散列索引相同。但是本发明并不限于此,布隆过滤器用散列索引与散列表用散列索引可以使用不同值。

[0068] 本发明中所使用的布隆过滤器,虽然有可能在特定值不存在于特定集时判断为存在于特定集(称之为“假阳性(False Positive)”),但不会发生特定值存在于特定集时判断出不存在于特定集的情况。因此,当使用利用了本发明所提供的布隆滤波器的数据包分类方法时,可以防止因不必要的元组检索而导致检索性能降低。

[0069] 从如表 1 的示例规则集的结果可以知道,现有的元组剪枝算法将元组 (1, 1)、(1, 3) 及 (3, 3) 均作为检索对象,但本发明利用布隆过滤器而只将元组 (1, 3) 作为检索对象。从而,可以减少用于规则检索的外部存储器存取次数,因此可以提高用于数据包分类的

检索性能。

[0070] 另一方面,前述的实施例中说明的是对于每个元组生成两个散列索引的情况,但并不限于此,可以只使用一个散列索引,也可以使用三个以上的散列索引。即,与散列索引的数量无关,只要使用与前述的数据包分类方法相同至均等的方法,都可以认为是被包括在本发明的技术思想中。

[0071] 以下的表 3 是使用了 3 个散列索引的情况的示例。

[0072] 【表 3】。

[0073]

| Rule No. | Source Prefix | Dest. Prefix | Tuple  | Concate. String | CRC Code | BF index   |
|----------|---------------|--------------|--------|-----------------|----------|------------|
| R0       | 1*            | 1*           | (1, 1) | 11              | 01101001 | 13, 26, 9  |
| R1       | 1*            | 110*         | (1, 3) | 1110            | 10000010 | 16, 0, 2   |
| R2       | 01*           | 00*          | (2, 2) | 0100            | 01101100 | 13, 27, 12 |
| R3       | 0*            | 111*         | (1, 3) | 0111            | 00000101 | 0, 1, 5    |
| R4       | 111*          | 110*         | (3, 3) | 111110          | 00010100 | 2, 5, 20   |
| R5       | 010*          | 110*         | (3, 3) | 010110          | 10101111 | 21, 11, 15 |

[0074] 布隆滤波器的大小定义为 M,要存储于布隆滤波器的要件(element)的数定义为 N,使用于编程的散列函数的数定义为 k 时,可以知道优化的散列函数的数如【公式 1】(Sarang Dharmapurikar, Praveen Krishnamurthy, and David E. Taylor, "Longest prefix matching using bloom filters," IEEE/ACM Transactions on Networking, vol. 14, no. 2, pp. 397-409, April 2006)。

[0075]  $k = \frac{M}{N} \ln 2$  ----- 【公式 1】

[0076] 因此,利用不同元组的数量(N)即 4 的 4 倍数,假设布隆滤波器的大小(M)为 16 比特时,用于编程布隆滤波器的最佳散列函数的数量为根据上述【公式 1】而决定为 3 个。表 3 示意的是在 8 比特 CRC 生成器的输出值中,使用 [7:4]、[5:2]、[3:0] 比特而生成的散列索引。例如,当 R0 的情况下,8 比特 CRC 生成器的输出值即,CRC 码 "01101001" 中,[7:4] 比特为 "0110", [5:2] 比特为 "1010", [3:0] 比特为 "1001",因此散列索引为 6、10、9。

[0077] 以下,对实现了根据本发明实施例的利用了布隆滤波器的数据包分类方法的数据包分类装置进行说明。

[0078] 图 3 为根据本发明实施例的利用了布隆滤波器的数据包分类装置的示意图。

[0079] 根据本发明实施例的利用了布隆滤波器的数据包分类装置 300 包括:元组生成部 310;数据包分类控制部 320;及存储器 330。

[0080] 元组生成部 310 是基本用于执行上述数据包分类方法中的(a)元组列表生成步骤及(b)元组挑选步骤的组件(module)。即,对输入数据包的源前缀字段与目标前缀字段分别进行字段别检索,并判断出与规则集相对应的字段的字段值的一致长度,从而生成由各字段的一致长度的组合构成的元组列表,并从生成的元组列表中只挑选出存在于规则集中的元组。

[0081] 其中,元组生成部 310 所执行的逐字段检索可以使用一维检索中被传为性能最佳的二分法检索级别(Binary Search on Level),但并不限于此。

[0082] 数据包分类控制部 320 是基本用于执行上述数据包分类方法中的 (c) 利用布隆滤波器的过滤步骤及 (d) 规则检索步骤的组件。即,对于元组生成部 310 所挑选的元组执行利用了布隆滤波器的过滤,并只将过滤结果为阳性的元组作为检索对象而存取存储于存储器 330 的散列表(hash tables),从而检索及决定出存储于散列表的规则中的最佳匹配规则。对于数据包分类控制部 320 的更加详细的构成,稍后参照图 4 及图 5 进行说明。存储器 330 中存储有具备了用于数据包分类的规则的散列表。图 4 为根据本发明第一实施例的数据包分类控制部的构成示意图。

[0083] 根据本发明的第一实施例的数据包分类控制部 320 包括散列部 410、布隆滤波器 420、及规则检索部 430。

[0084] 散列部 410 对于元组生成部 310 所挑选出来的每个元组生成出散列索引,可以包括 CRC 生成器 412 及散列索引生成部 414。

[0085] CRC 生成器 412,被输入字符串而生成固定长度的散列码。所述字符串是从输入数据包的源前缀及目标前缀中取出元组生成部 310 所挑选出来的元组所表示的一致长度大小的较高序位比特,并相互串接(Concatenation)而生成的。

[0086] 散列索引生成部 414 根据 CRC 生成器 412 所生成的固定长度散列码而生成散列索引。散列索引生成部 414 可以从固定长度散列码取出一定比特(例如,几个最高有效比特或/及几个最低有效比特)而生成散列索引,并且可以生成一个或多个散列索引。布隆滤波器 420 确认散列索引生成部 414 所生成的散列索引所指的条目的比特值后,只挑选比特值为阳性的元组作为检索对象。

[0087] 规则检索部 430 存取布隆滤波器 420 中作为检索对象而挑选出来的元组的散列索引所指的散列表的条目后,执行规则检索,并决定出与输入数据包一致的规则中的最佳匹配规则。

[0088] 图 5 为根据本发明第二实施例的数据包分类控制部的构成示意图。

[0089] 根据本发明第二实施例的数据包分类控制部 320 的散列部 510,分别生成布隆滤波器用散列索引与散列表用散列索引,这点与第一实施例不同。

[0090] 即,散列索引生成部 514 利用由 CRC 生成器 512 所生成的固定长度散列码,分别生成布隆滤波器用散列索引与散列表用散列索引。其中,布隆滤波器用散列索引可以是一个或者多个,散列表用散列索引是单个。

[0091] 布隆滤波器 520 确认布隆滤波器用散列索引所指的条目的比特值后,只挑选出比特值为阳性的元组作为检索对象,规则检索部 530 存取作为检索对象而挑选的元组的散列表用散列索引所指的散列表条目,并检索及决定出与输入数据包一致的规则中的最佳匹配规则。

[0092] 以上说明的本发明的一实施例中,将元组所表示的一致长度大小的较高序位比特分别从源前缀与目标前缀中取出后相互串接而成的字符串,作为用于生成散列索引的散列函数(CRC 生成器)的输入值而使用。但是,这只是一个实施例,本发明的思想并不局限于此,作为用于生成散列索引的散列函数的输入值,可以使用各种输入值。

[0093] 例如,作为减少利用布隆滤波器而进行过滤时所发生的假阳性(False Positive)的数量的方法,从源前缀与目标前缀中分别取出元组所表示的一致长度大小的较高序位比特,给取出来的值加上所定的位模式(bit pattern)而填补(Padding)至固定长度值后,被

填补的值相互串接而得到的值可以作为散列函数的输入值而使用。

[0094] 例如,从源前缀取出的值设为“A”时,反复使用将3比特的“101”作为后缀(trailer)而添加于A后面的“A101”模式,从而扩张大至32比特,并以相同方式对从目标前缀中取出的B扩大至32比特后,可以将这些相互串接的64比特的值作为散列函数的输入值。即使输入数据包与规则不一致,只要散列函数的输入值相同就会生成相同的散列索引,因此会发生利用布隆滤波器的过滤结果变成假阳性的情况。

[0095] 例如,假设被存储的规则为(1\*,\*),分别从源前缀与目标前缀取出元组所表示的一致长度大小的较高序位比特,再相互串接而形成字符串并作为散列函数的输入值而使用的实施例作为基础时,散列函数的输入字符串为‘1’。假设输入为(1100,1000),而逐字段检索结果得到的是元组(0,1)时,用于得到输入所相应的散列索引的散列函数的输入字符串为‘1’,从而会得到与之前存储的规则所相同的散列索引。但实际被存储的规则与输入的元组不一致,因此可以知道所相应的输入的元组(0,1)为假阳性。作为另一个实例,假设被存储的规则为(1\*,0\*)时,该规则的散列函数输入字符串为‘10’,并以与前述的实例相同的逐字段检索结果而得到元组(0,2)的情况下,相应的输入的散列函数输入字符串也为‘10’,从而得到与被存储的规则相同的散列索引。这种情况下,相应的输入的元组与值均与被存储的规则不同,但由于得到相同的散列索引而导致假阳性。

[0096] 使用填补的方法能提高不同的元组生成不同散列函数的输入值的可能性,因此降低发生假阳性的频率。从而,可以减少用于检索最佳匹配规则的存储器存取次数。

[0097] < 本发明的性能评价 >

[0098] 使用了Classbench,对于大概具有5000个规则的ACL、IPC、FW集来对本发明的实施例中所提出的结构性能进行了评估。所述Classbench的特性与骨干路由器实际使用的规则集的特性相似。

[0099] 表4涉及的是使用于试验的各规则集的特性,示意了规则集所包含的规则的数量、不同的元组的数量、及不同的字段值的组合数量。ACL5K规则集所包含的元组种类少于其它规则集,因此不同的元组的数量少,但是一个元组内具有不同前缀值的组合的规则较多,因此特征是具有不同字段值的组合的数量较大。相反地,FW5K的元组种类虽然多样,但前缀组合相同的规则较多,因此特征是不同字段值的组合的数量少于其它规则集。

[0100] 【表4】。

[0101]

| Rule Set | No. of Rules | No. of Distinct Tuples | No. of Distinct Combinations<br>in Prefix Values |
|----------|--------------|------------------------|--------------------------------------------------|
| ACL 5K   | 4660         | 102                    | 2453                                             |
| IPC 5K   | 4486         | 680                    | 2933                                             |
| FW 5K    | 4351         | 579                    | 1244                                             |

[0102] 表5示意了采用了元组布隆滤波器的元组剪枝算法的性能,是对如下将元组所表示的一致长度大小的较高序位比特分别从源前缀与目标前缀取出后相互连接,并将由此生成的字符串作为散列函数的输入值而使用的实施例的性能评估结果。

[0103] 【表 5】。

[0104]

| Rule Set | No. of Inputs | $T_{Pruned}$ | Tuple Bloom Filter |                |                    |                              |                           | Tuple Pruning Rate by Tuple BF (%) | True Match ( $T_{true\_match}$ ) |
|----------|---------------|--------------|--------------------|----------------|--------------------|------------------------------|---------------------------|------------------------------------|----------------------------------|
|          |               |              | $M$                | No. of Indices | $T_{Neg}$ Negative | Positive( $T_{BF\_Pruned}$ ) |                           |                                    |                                  |
|          |               |              |                    |                |                    | True ( $T_{BF\_1-field}$ )   | False ( $T_{BF\_false}$ ) |                                    |                                  |
| ACL 5K   | 13980         | 6.87         | $N'$               | 1              | 1.79               | 3.87                         | 1.22                      | 26.01                              | 3.29                             |
|          |               |              | $2N'$              | 1              | 2.36               |                              | 0.65                      | 34.28                              |                                  |
|          |               |              | $4N'$              | 3              | 2.90               |                              | 0.11                      | 42.12                              |                                  |
|          |               |              | $8N'$              | 6              | 2.96               |                              | 0.04                      | 43.11                              |                                  |
|          |               |              | $16N'$             | 11             | 2.96               |                              | 0.04                      | 43.12                              |                                  |
|          |               |              | $32N'$             | 22             | 2.94               |                              | 0.07                      | 42.73                              |                                  |
| IPC 5K   | 20610         | 15.29        | $N'$               | 1              | 3.77               | 5.41                         | 6.11                      | 24.62                              | 4.00                             |
|          |               |              | $2N'$              | 1              | 4.95               |                              | 4.93                      | 32.35                              |                                  |
|          |               |              | $4N'$              | 3              | 5.78               |                              | 4.10                      | 37.81                              |                                  |
|          |               |              | $8N'$              | 6              | 5.95               |                              | 3.93                      | 38.89                              |                                  |
|          |               |              | $16N'$             | 11             | 5.95               |                              | 3.93                      | 38.90                              |                                  |
|          |               |              | $32N'$             | 22             | 5.85               |                              | 4.03                      | 38.26                              |                                  |
| FW 5K    | 13054         | 8.37         | $N'$               | 1              | 0.54               | 6.14                         | 1.70                      | 6.39                               | 5.82                             |
|          |               |              | $2N'$              | 1              | 0.92               |                              | 1.31                      | 11.03                              |                                  |
|          |               |              | $4N'$              | 3              | 1.08               |                              | 1.15                      | 12.92                              |                                  |
|          |               |              | $8N'$              | 6              | 1.10               |                              | 1.14                      | 13.08                              |                                  |
|          |               |              | $16N'$             | 11             | 1.05               |                              | 1.18                      | 12.50                              |                                  |
|          |               |              | $32N'$             | 22             | 1.10               |                              | 1.14                      | 13.08                              |                                  |

[0105] 该试验中将不同元组的数量设为  $N$  时, 元组布隆滤波器的大小单位  $N'$  设为  $2^{\lceil \log_2 N \rceil}$ 。散列函数使用了 CRC-64 生成器, 散列索引的数量 ( $k$ ) 随布隆滤波器的大小 ( $M$ ) 而增加。逐字段一维检索使用了被传性能最优秀的 Binary Search on Level (BSL)。表 5 中的  $T_{Pruned}$  表示只采用现有的元组剪枝算法时剩余的元组的平均数量。 $T_{BF\_Pruned}$  表示根据元组布隆滤波器大小的存储器存取性能, 根据的是: 元组布隆滤波器所筛选出来的平均阴性元组的数量 ( $T_{Neg}$ ); 在阳性结果中生成被存储的规则与元组时所使用的两个字段相一致的真正阳性元组的平均数量 ( $T_{BF\_2-field}$ ); 虽然通过了元组布隆滤波器但与被存储的规则并不一致的假阳性元组的平均数量 ( $T_{BF\_false}$ )。元组布隆滤波器剪枝比率 (Tuple Pruning Rate by

Tuple BF) 是以百分比表示元组布隆过滤器从现有的元组剪枝算法中进一步筛选出来的额外元组数,并提供对于一个输入所相应的各元组检索的结果中最终与被存储的规则完全一致的元组的平均数量 ( $T_{\text{true\_match}}$ )。

[0106] 通过表 5 的数据,对于通过元组剪枝算法而得到的结果即剩余的元组 ( $T_{\text{Pruned}}$ ),所提供的元组布隆过滤器进一步除去不存在于规则集的元组,由此可以知道实际存取存储器的元组的数量 ( $T_{\text{BF\_Pruned}}$ ) 减少了。而且,还可以知道布隆过滤器的大小 (M) 越大时得到阴性结果的比率增加,由此通过元组布隆过滤器的元组的数量 ( $T_{\text{BF\_Pruned}}$ ) 渐渐减少。布隆滤波器的特性即假阳性也同样受布隆过滤器大小 (M) 的影响,布隆过滤器的大小越大,则假阳性有减少的倾向。

[0107] 其中所谓的真正阳性 ( $T_{\text{BF\_2-field}}$ ) 是,在通过了布隆滤波器的阳性结果中,经存取散列表而与被存储的规则进行比较的结果,输入的源与目标前缀的值相一致的情况。真正阳性元组的平均数量是一定的,与布隆过滤器的大小无关,但假阳性元组的平均数量会随着布隆过滤器的大小而发生变化。

[0108] 从表 5 中可以知道,采用了元组布隆滤波器的元组剪枝算法执行附加的元组除去功能,从而可以减少外部存储器存取次数。但是,根据布隆滤波器的特性,随元组布隆过滤器的大小的增加而假阳性的数量应该减少,但从特定大小起,假阳性不再减少而是收敛。特别是在 IPC5K 或 FW5K 规则集的情况下,随布隆过滤器大小的增加,假阳性元组数量的减少的幅度非常小。

[0109] 为了减少这样的假阳性的数量,如前面所述,从源前缀与目标前缀分别取出元组所表示的一致长度大小的较高序位比特,并在取出的值上附加所定的位模式而填补 (Padding) 至固定长度值后,将连接所填补的值而所得到的值可以作为散列函数的输入值来使用,其结果如表 6 所示。

[0110] 【表 6】

[0111]

| Rule Set | No. of Inputs | $T_{Pruned}$ | Tuple Bloom Filter |                |          |                              |                           | Pruned Tuple Ratio by Tuple BF (%) | True Match ( $T_{true\_match}$ ) |
|----------|---------------|--------------|--------------------|----------------|----------|------------------------------|---------------------------|------------------------------------|----------------------------------|
|          |               |              | $M$                | No. of Indices | Negative | Positive( $T_{BF\_Pruned}$ ) |                           |                                    |                                  |
|          |               |              |                    |                |          | True ( $T_{BF\_2-field}$ )   | False ( $T_{BF\_false}$ ) |                                    |                                  |
| ACL 5K   | 13980         | 6.875        | $N'$               | 1              | 1.707    | 3.868                        | 1.301                     | 24.82                              | 3.29                             |
|          |               |              | $2N'$              | 1              | 2.313    |                              | 0.693                     | 33.65                              |                                  |
|          |               |              | $4N'$              | 3              | 2.918    |                              | 0.089                     | 42.44                              |                                  |
|          |               |              | $8N'$              | 6              | 3.007    |                              | 0.000                     | 43.74                              |                                  |
|          |               |              | $16N'$             | 11             | 3.007    |                              | 0.000                     | 43.74                              |                                  |
|          |               |              | $32N'$             | 22             | 3.007    |                              | 0.000                     | 43.74                              |                                  |
| IPC 5K   | 20610         | 15.290       | $N'$               | 1              | 4.979    | 5.411                        | 4.901                     | 32.56                              | 4.00                             |
|          |               |              | $2N'$              | 1              | 5.811    |                              | 4.070                     | 38.00                              |                                  |
|          |               |              | $4N'$              | 3              | 9.013    |                              | 0.867                     | 58.95                              |                                  |
|          |               |              | $8N'$              | 6              | 9.775    |                              | 0.105                     | 63.93                              |                                  |
|          |               |              | $16N'$             | 11             | 9.778    |                              | 0.102                     | 63.95                              |                                  |
|          |               |              | $32N'$             | 22             | 9.778    |                              | 0.102                     | 63.95                              |                                  |
| FW 5K    | 13054         | 8.374        | $N'$               | 1              | 0.618    | 6.142                        | 1.613                     | 7.39                               | 5.82                             |
|          |               |              | $2N'$              | 1              | 1.232    |                              | 0.999                     | 14.71                              |                                  |
|          |               |              | $4N'$              | 3              | 1.835    |                              | 0.396                     | 21.92                              |                                  |
|          |               |              | $8N'$              | 6              | 1.916    |                              | 0.315                     | 22.88                              |                                  |
|          |               |              | $16N'$             | 11             | 1.825    |                              | 0.306                     | 22.99                              |                                  |
|          |               |              | $32N'$             | 22             | 1.925    |                              | 0.306                     | 22.99                              |                                  |

[0112] 在表 6 中,使用了将各字段的前缀扩大到固定长度 32 比特的填补。这时填补的长度与值可以任意决定,在本试验中,将 {prefix, 101} 模式反复填补至 32 比特,所述 {prefix, 101} 模式即为,将 3 比特的 '101' 作为字段后缀(trailer)而加在各字段的前缀后面的模式。如表 6 中可以知道,通过将散列函数的输入均相同地扩大至 64 比特而得到的索引,对布隆滤波器进行编程并进行检索的结果,当布隆滤波器的大小为  $4N'$  时整个规则集的假阳性元组的数量低于 1 个。

[0113] 图 7 及图 8 为现有的元组剪枝算法与根绝本发明实施例的数据包分类方法之间的存储器存取次数相关的性能比较图。

[0114] 图 7 中将本发明实施例中未使用填补的情况下与现有的元组剪枝算法的比较结果。

[0115] 为了各算法的性能评价而使用的散列函数 64 比特 CRC 函数, 不管元组的种类均可以得到相同长度的散列码, 因此优点是可以得到特定长度的散列索引, 但由于不是完美的散列函数, 因此会发生对不同的输入发生相同的散列索引这样的冲突。在性能评价试验中发生散列索引的冲突的时连接至链表(linked-list)后进行存储, 并在检索时对于通过链表(linked-list)而被存储的条目进行线性检索。通过图 7 可以知道, 在现有的元组剪枝算法中仅用前缀长度数据的组合而成的元组来限制检索领域的基础上, 所提出的结构进一步使用大小较小(2 ~ 4Kbyte)的片上存储器(on-chip memory)的布隆滤波器, 对前缀值数据再次限制检索领域, 使得可以有效地实现更快的数据包分类。图 8 中, 将本发明实施例中使用了填补的情况下与现有的元组剪枝算法的比较结果。

[0116] 其中, 图 8(a) 为假设散列函数为不发生冲突的完美散列函数时的比较图表, 如图 8(a) 所示, 可以知道外部存储器存取次数可以最多减少 40%。

[0117] 另一方面, CRC 函数如上所述并不是完美的散列函数, 因此在发生散列索引的冲突时, 通过链表(linked-list)来连接后进行存储, 在检索时对于通过链表(linked-list)而存储的条目进行线性检索, 其结果图示于图 8(b)。

[0118] 参照图 8(b), 可以知道与假设完美的散列函数的图 8(a) 相比, 其性能更加提高。

[0119] 而且, 将不使用填补的图 7 与使用了填补的图 8(b) 进行比较时, 可以知道使用了填补的情况下更加提高性能的增加。

[0120] 虽然在以上的说明中, 构成本发明实施例的所有构成要件结合为一个或结合后运行, 但本发明并不被这样的实施例限制。即, 只要是在本发明的目的范围内, 其所有构成要件可以有选择性地被结合的一个以上并运行。而且, 其所有构成要件可以分别体现成一个独立的硬件, 但也可以体现为具有程序模块的电脑程序, 被有选择性地组合各构成要件的其部分或者全部, 从而执行在一个或多个硬件中被组合的部分或者全部功能。构成该电脑程序的代码及代码段会轻易被本发明所属技术领域的技术人员推断出来。这样的电脑程序被存储于电脑能读取的存储媒体(Computer Readable Media)后被电脑读取并实施, 从而可以实现本发明的实施例。作为电脑程序的存储媒体, 可以包括磁记录媒体、光记录媒体、载波媒体等。

[0121] 而且, 以上记载的“包括”、“构成”、“具有”等术语的意思是, 只要没有特别反对的记载, 就可以内置相应构成要件, 因此并不是排除其它构成要件, 而是应该解释为可以包括其它构成要件。包括了技术上或科学上的术语的术语, 只要不是被定义成不同的意思, 在本发明所属的技术领域中具有通常知识的技术人员在一般情况下所理解的意思相同。如词典中被定义的术语, 常被使用的术语应该被解释为与技术的文脉上的意思相一致, 只要本发明中没有明确地进行定义, 就不应被解释为异常或过于形式的意思。

[0122] 以上的说明只不过是用于举例说明本发明技术细想的实施例, 在本发明所属的技术领域中具有通常知识的技术人员可以在不超出本发明的本质特性的范围内进行各种修改及变换。从而, 在本发明所展示的实施例是用来说明本发明的技术思想的, 并不是用来限定的, 并不是通过这样的实施例来限定本发明的技术思想的范围。本发明的保护范围应通过如下的权利要求范围来进行解释, 与其有等同范围的所有技术思想都应该被解释为包括在本发明的权利范围内。

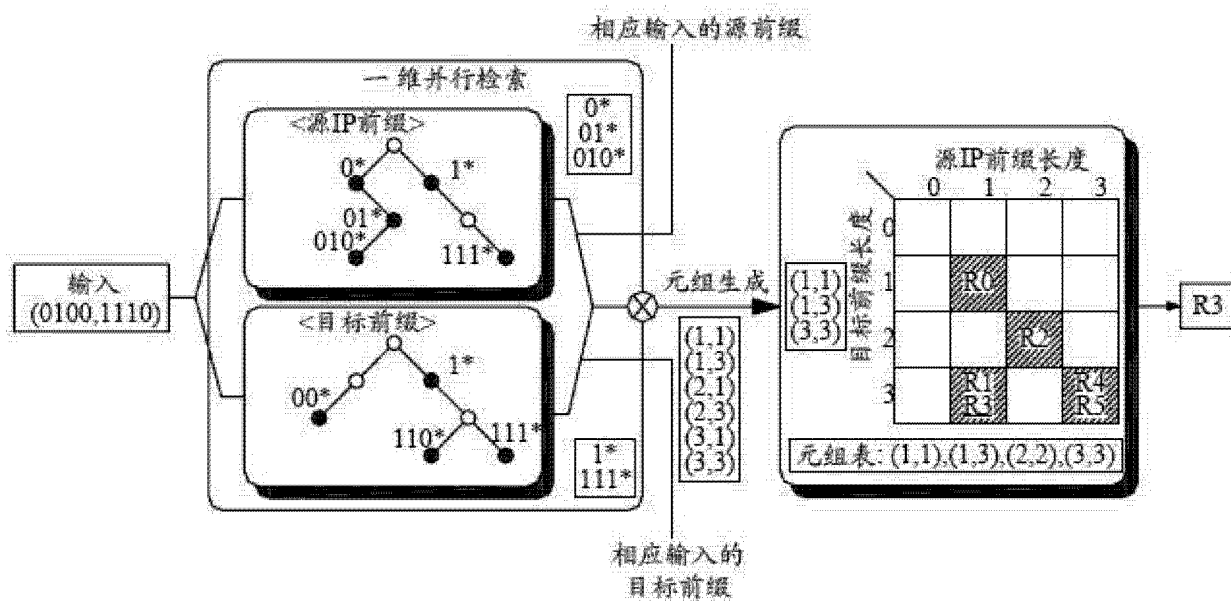


图 1

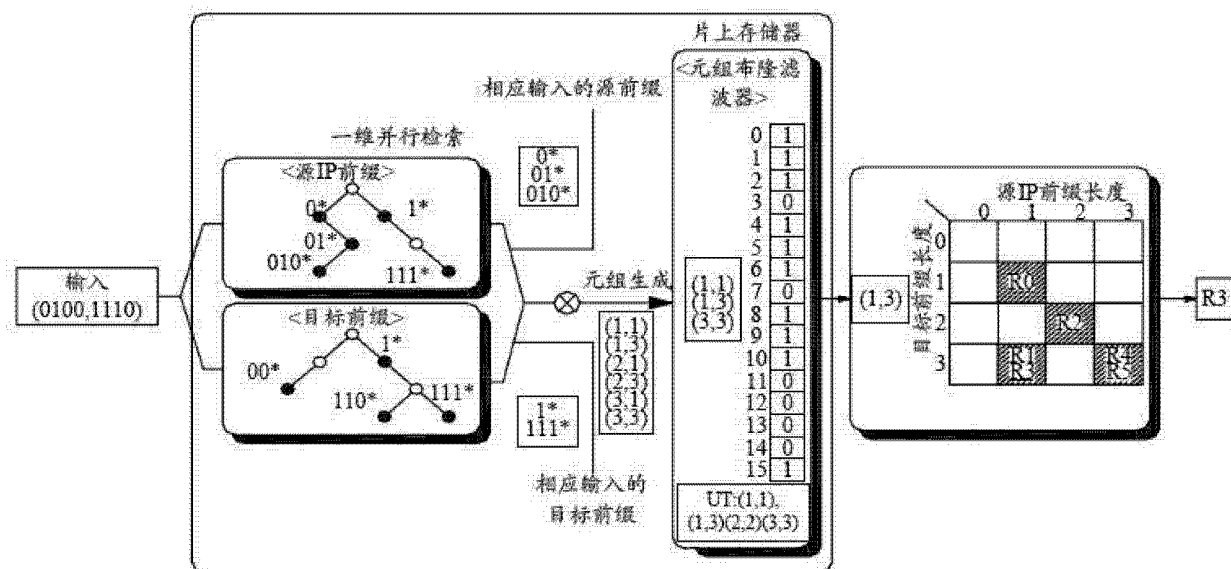


图 2

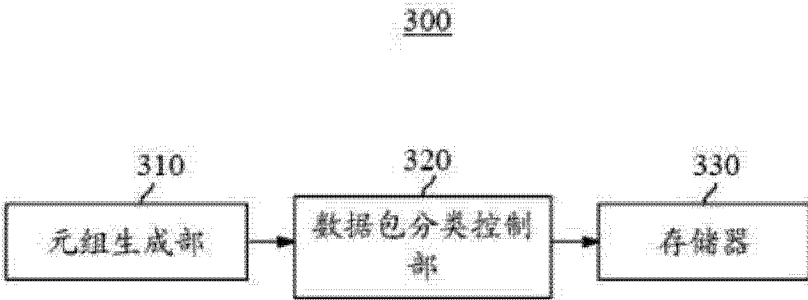


图 3

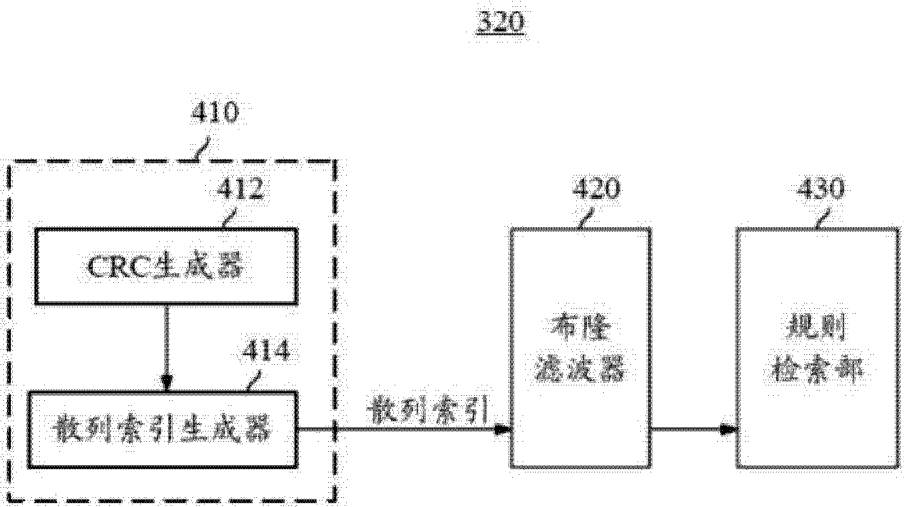


图 4

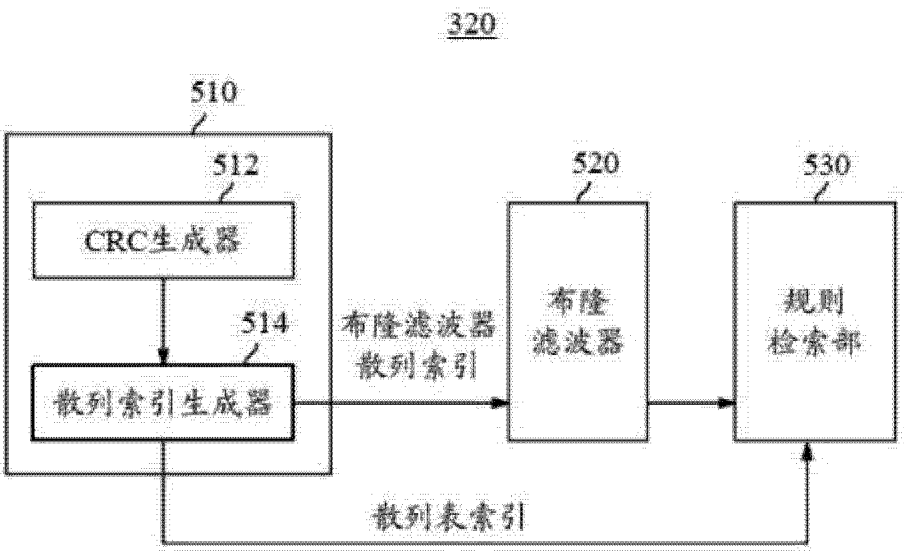


图 5

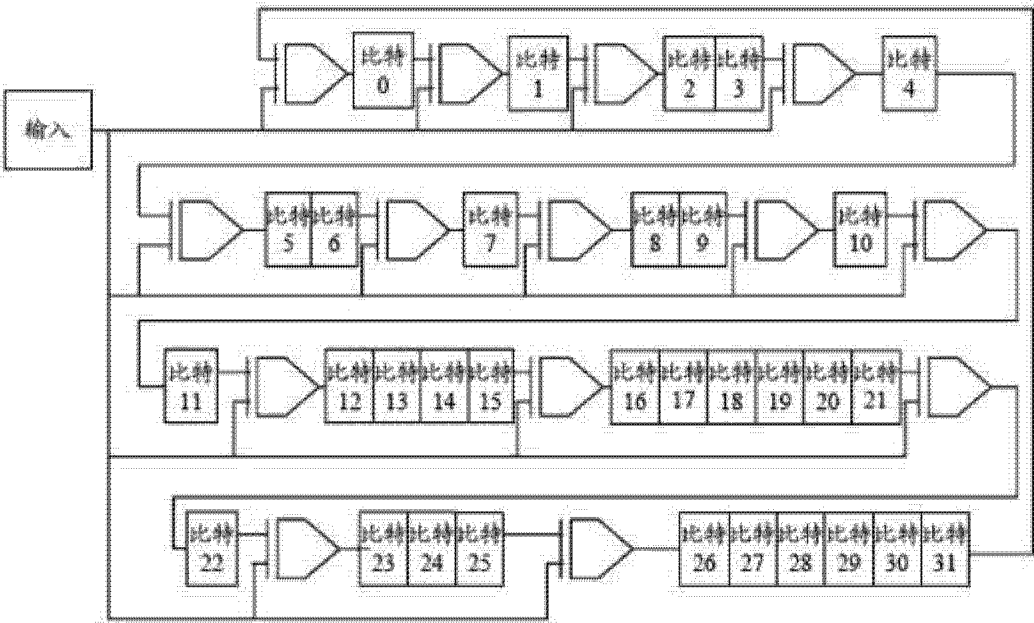


图 6

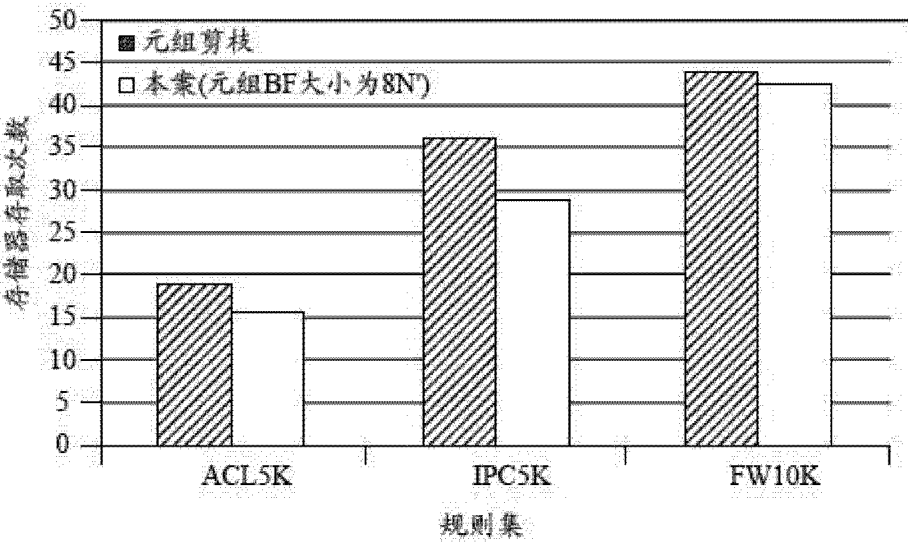


图 7

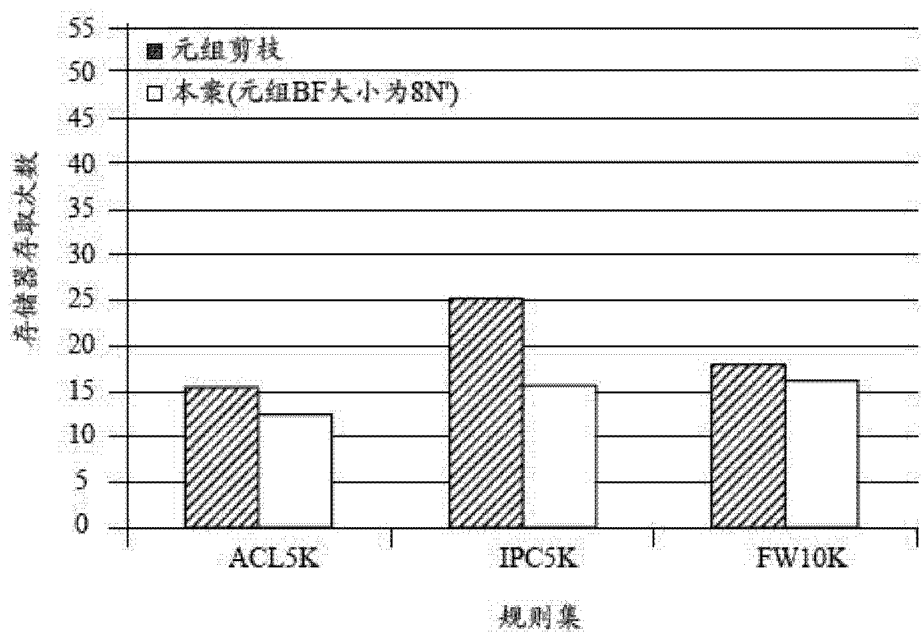


图 8A

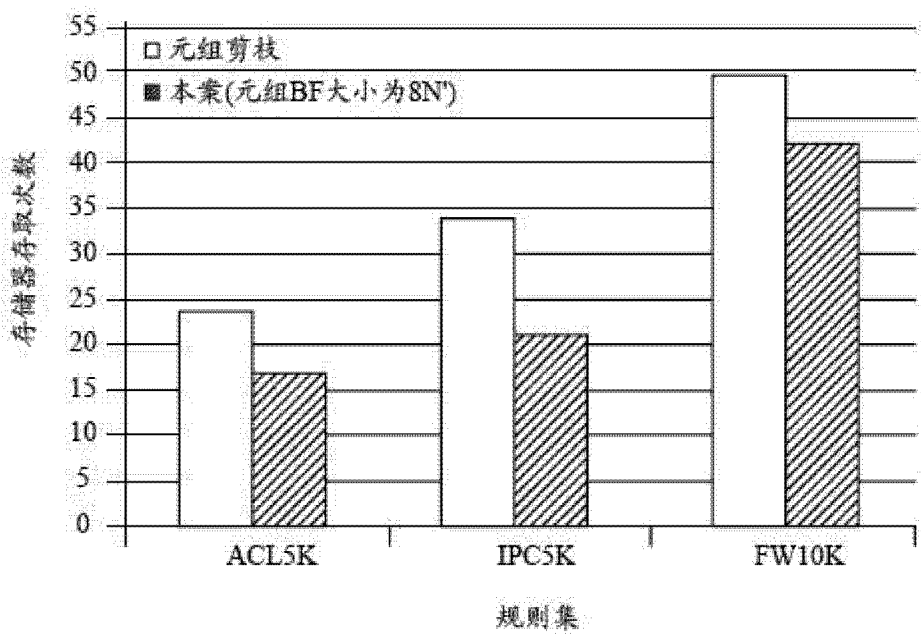


图 8B