

基于并行 Bloom 过滤器组的深度数据包检测算法

胡国良, 林亚平*, 王 刚, 姚 鑫

(湖南大学 信息科学与工程学院, 长沙 410082)

(* 通信作者电子邮箱 yplin@hnu.edu.cn)

摘 要: 针对基于软件、硬件的深度数据包检测存在处理速度慢或规则集更新困难等方面的局限性, 提出一种在多核平台上基于并行 Bloom 过滤器组的深度数据包检测算法。算法中首先将规则集按规则的长度分组, 构造一个并行 Bloom 过滤器组, 组中每个计数式 Bloom 过滤器表示特定规则长度的规则集。为了减少执行过程中的冲突概率和计算量, 构造了高性能的哈希函数, 然后基于多核平台的并行处理能力使用并行编程实现了该算法。理论分析和实验结果表明该算法是一种时空高效的算法。

关键词: 深度数据包检测; 规则集; 多核平台; 计数式 Bloom 过滤器; 并行 Bloom 过滤器组

中图分类号: TP393.08 **文献标志码:** A

Deep packet inspection algorithm based on parallel Bloom filters

HU Guo-liang, LIN Ya-ping*, WANG Gang, YAO Xin

(College of Information Science and Engineering, Hunan University, Changsha Hunan 410082, China)

Abstract: The traditional methods for hardware/software-based Deep Packet Inspection (DPI) have intrinsic limitations in practical implementation. To address the shortcomings, this paper presented a DPI algorithm implemented on the multi-core platform based on the parallel Bloom filters. Firstly, the algorithm grouped the rule sets according to their lengths and constructed a set of counting Bloom filters to represent the grouped rule sets. Each Bloom filter stood for a rule set with a specific length. Secondly, efficient hash functions were introduced to reduce the collision probability and the computing complexity. Lastly, the algorithm was implemented using the parallel programming method based on the parallel processing ability of the multi-core platform. The theoretic analysis and experimental results show that the proposed algorithm is time and space efficient.

Key words: deep packet inspection; rule set; multi-core platform; Counting Bloom Filter (CBF); parallel Bloom filters

0 引言

随着网络技术的高速发展和广泛应用, 诸如网络蠕虫、计算机病毒、僵尸网络等新型的入侵和劫持敏感隐私信息的攻击也层出不穷^[1-3]。如何高速检测并阻止这些入侵和攻击行为引起了研究学者的广泛关注。

深度数据包检测 (Deep Packet Inspection, DPI) 是网络入侵检测系统 (Network Intrusion Detection System, NIDS) 与网络入侵防护系统 (Network Intrusion Prevention System, NIPS) 的核心^[3], 它不仅检测数据包头部信息而且可监测数据包的有效载荷, 通过将数据包的内容与一组预先定义的规则进行匹配来监测和阻止具有危害行为的网络数据包, 能够弥补传统安全防护体系无法解决应用层攻击的不足。

如何设计高速有效的 DPI 监测引擎是一个具有挑战性的课题: 1) 由于 DPI 技术在 NIDS/NIPS 系统中应用的实时性, 要求监测引擎能够以线速的吞吐量实时处理每一个网络数据包, 因此, 如何设计高速的 DPI 监测引擎是一个具有挑战性的问题; 2) 近年来网络中的攻击手段和攻击方式层出不穷, 对应的监测规则也不断增加, 如何设计有效的数据结构存储大

量的特征规则是另一个挑战所在。

目前的 DPI 研究工作主要分为基于软件和基于硬件的实现。基于软件的 DPI 技术, 如克努思—莫锐斯—帕罗特 (Knuth-Morris-Pratt, KMP) 算法^[4]、贝叶—莫尔 (Boyer-Moore, BM) 算法^[5] 及阿霍—克若思克 (Aho-Corasick, AC) 算法^[6] 等, 主要面向单核平台设计串行特征匹配算法。但随着网络带宽和业务流量的迅猛增长以及特征规则集的日益增多, 基于这类算法的 DPI 已无法满足线速数据包处理的高性能需求。基于硬件的监测引擎主要包括: 基于三重内容可寻址存储器 (Ternary Content Addressable Memory, TCAM)、基于现场可编程门阵列 (Field Programmable Gate Array, FPGA) 和基于多核平台的技术^[7]。

基于 TCAM 的包过滤技术^[8-9] 利用 TCAM 查找速度快、操作简单等优点, 满足了高速网络中数据包线速处理需求, 但 TCAM 相对其他存储介质而言价格昂贵, 存储芯片的容量相对较小, 存储效率不高; 其次, 由于 TCAM 是基于并行的匹配方式, 功耗较大, 查找过程中需要对所有规则都进行比较, 因而造成了大量的比较浪费; 最后, 其需要保证前缀较长的规则保存在前缀较短的规则之前, 这种特定的顺序关系使得规则

收稿日期: 2012-05-13; 修回日期: 2012-06-25。

基金项目: 国家自然科学基金资助项目 (60973031); 湖南省研究生科研创新项目 (CX2011B138)。

作者简介: 胡国良 (1985 -), 男, 湖南娄底人, 硕士研究生, 主要研究方向: 高性能网络; 林亚平 (1955 -), 男, 湖南邵阳人, 教授, 博士, 主要研究方向: 通信网络、机器学习; 王刚 (1985 -), 男, 甘肃天水人, 博士研究生, 主要研究方向: 高性能网络、接入控制; 姚鑫 (1989 -), 男, 湖南岳阳人, 硕士研究生, 主要研究方向: 高性能网络。

更新变得复杂。

文献[10]提出了一种在 FPGA 上的基于并行 Bloom 过滤器的深度数据包检测算法。该方法结合了 FPGA 的设计周期短、设计灵活且速度快等特点以及 Bloom 过滤器空间简洁、查询方便的优点,获得了比较好的实验结果,但该算法要求高吞吐量的并行存储器访问带宽,存在代价昂贵和可伸缩性差等问题^[3]。同时加上 FPGA 本身存在规则集更新困难、片上内存较少、不能实现复杂的逻辑等不足,使得基于 FPGA 的实现在规则集日益增多、规则更新频繁的高速网络中面临着挑战。

文献[7]提出了在多核平台上设计与实现基于硬件的高性能特征匹配方法,已吸引越来越多研究者的关注。多核处理器是在一块芯片上集成多个 CPU 核,具有高速灵活的并行计算能力,随着多核处理器等硬件技术的不断发展,例如 Intel Xeon CPU 和 GPGPU 集成了成百上千个核,为 DPI 的性能提升提供了新的机遇与挑战^[3]。

本文利用多核架构的特点,结合 Bloom 过滤器空间简洁、查询方便的特征,提出一种在多核架构上基于并行 Bloom 过滤器组的深度数据包检测算法,该算法首先将预先定义的规则集按规则的长度分组,然后构造并行 Bloom 过滤器组,组中每个计数式 Bloom 过滤器表示特定规则长度的规则集。为了减少冲突率和计算量,构造了高效的哈希函数,然后基于多核平台的并行处理能力使用并行编程实现了该算法。本文采用的方法与已有的基于软件的工作相比,提高了匹配吞吐量,降低了存储空间;与基于硬件的工作相比,编程灵活,规则集更新容易,是一种时间和空间有效的算法。

1 问题描述与模型建立

本章首先介绍涉及到 Bloom 过滤器的一些基本概念,在此基础上提出了一种基于并行 Bloom 过滤器的深度数据包检测模型。

1.1 Bloom 过滤器

Bloom 过滤器是一种空间高效的数据结构,能够高效精确地表示信息^[11]。

标准 Bloom 过滤器的算法模型如图 1 所示,规则集合 $S = \{s_1, s_2, \dots, s_n\}$ 共有 n 个元素通过 k 个哈希函数 $h_1(x), h_2(x), \dots, h_k(x)$ 映射到长度为 m 初始值全为 0 的位串数组 V 中,一个元素仅占用几个比特位且其所需的存储空间与元素自身的大小及规则集合的规模无关,极大地节约了存储空间。但 Bloom 过滤器存在假阳性误判(即将不属于集合的元素误判为属于集合),其产生假阳性的概率 f 可以表示为:

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

其中 n 为规则集 S 的大小。根据式(1)式可知恰当地选择 m 值和 k 值可以降低假阳性概率 f ,由式(1)可知当 k 与 m/n 的关系为:

$$k = (m/n) \ln 2 \quad (2)$$

此时相应的最小假阳性概率为:

$$f = (1/2)^k \quad (3)$$

m/n 的值表示 Bloom 过滤器映射存储每个元素所需要的比特位。由式(2)、(3)可知:随着 m/n 值的增大, k 值会相应线性增长,从而使得假阳性概率降低。

由于标准 Bloom 过滤器中的 V 数组所有位被集合元素所共享,不支持元素的删除。为了支持集合动态变化时的删除

操作,文献[12]提出了计数式 Bloom 过滤器(Counting Bloom Filter, CBF)。采用计数器替代标准 Bloom 过滤器 V 中的比特位,如图 2 所示。

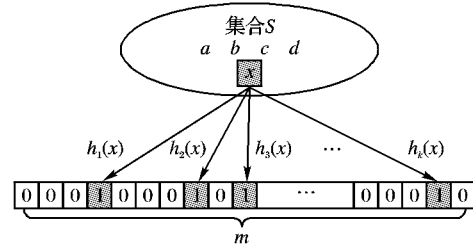


图 1 标准 Bloom 过滤器

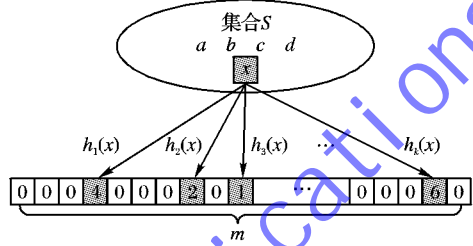


图 2 计数式 Bloom 过滤器

计数式 Bloom 过滤器中计数器的引入虽然增大了算法的空间占用率,但提供了元素的删除操作,解决了元素动态删除的问题,且相对于其他算法而言,其空间占用率仍然很低。

1.2 基本模型

本节提出一种基于并行 Bloom 过滤器组的深度数据包检测模型。

规则集 S 是攻击特征库,每条规则都对应一条攻击特征,令规则长度为 l 字节, $L = l_{\max} - l_{\min}$,其中 l_{\min} 为最小规则长度, l_{\max} 为最大规则长度, $l \in [l_{\min}, l_{\max}]$ 。对给定 S 首先按规则的长度分组成 L 个规则子集 $S_1, S_2, \dots, S_i, \dots, S_L$,然后由 L 个计数式 Bloom 过滤器 $CBF_1, CBF_2, \dots, CBF_L$ 构造并行 Bloom 过滤器组,每个 CBF 仅表示特定规则长度的规则集 $S_i (i \in [1, L])$ 。

基本模型如图 3 所示,该模型主要分为两部分:并行 Bloom 过滤器组和分析器。并行 Bloom 过滤器组监视网络数据流并从中搜索、匹配预先定义的规则集,一旦发现网络中的流量特征子串属于自身成员时,则将该子串标记为可疑子串并交由分析器;分析器用来消除可疑子串中属于假阳性误判的子串,根据判断结果作出相应处理(过滤,放行)。



图 3 基于并行 Bloom 过滤器的深度数据包检测模型

设定并行 Bloom 过滤器组监视的数据流窗口长度为 L_{\max} ,如图 4 所示,当网络数据流填满整个窗口时,它包含 L 个不同长度的子字符串,这些子字符串的长度介于规则长度范围之内,都有可能属于规则集成员,因而都被并行地分别交由相应的 CBF 进行匹配检测,匹配上的子串被标记为可疑子串并交由分析器进行假阳性误判分析。如果该过程出现多个可疑子串,则分析器从最长可疑子串到最短可疑子串依次进行假阳性误判分析,当遇到某个可疑子串属于规则集成员时就停止后续可疑子串的分析,该可疑子串就是此次并行过程中的敏

感字符串,最后分析器对敏感字符串进行过滤,对其他可疑子串则直接放行。

并行 Bloom 过滤器组通过上述方式监视数据流窗口,每个数据包中所有长度从 L_{\min} 到 L_{\max} 的子串都被扫描,同时由于并行 Bloom 过滤器组是并行工作的,因而这些子字符串可以在一次并行执行过程中匹配检测完。如存在可疑子串则交由分析器做进一步的分析处理,然后数据流窗口向右移动一个字节;如不存在可疑子串,则数据流窗口直接向右移动一个字节,重复上述操作。

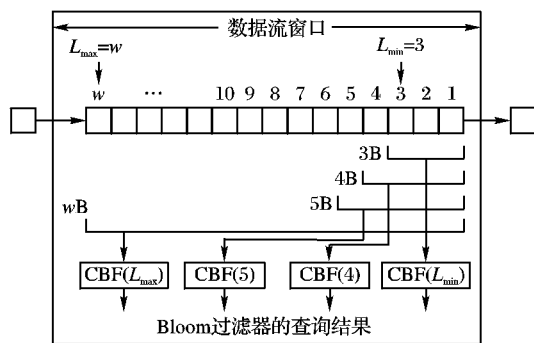


图4 Bloom过滤器组的数据流窗口

2 算法设计

本章实现了基于并行 Bloom 过滤器组的深度数据检测算法,构造了高性能的哈希函数,设置了计数式 Bloom 过滤器参数。

2.1 Bloom 过滤器组的并行算法

传统基于单核 CPU 平台上设计与实现的串行程序无法满足解决大规模问题的需求,而多核处理器等硬件技术在不断发展且具有强大的并行计算能力,本节提出了在多核平台上计数式 Bloom 过滤器组的并行算法。

当计数式 Bloom 过滤器组并行地从数据窗口中读取子串进行匹配检测时,各个计数式 Bloom 过滤器之间的执行过程是相互独立的,因而可以非常方便地利用多核的并行计算能力来实现算法的并行化,并行计数式 Bloom 过滤器组的并行算法具体描述如下:

```

Input: Bloom[ L ]
Begin
  分配 L 个处理器 P1, P2, ..., PL 并分别设置一个标记位;
  For i = 1 to L do in parallel
    为下标 i-1 的计数式 Bloom 过滤 Bloom[ i-1 ] 分配处理器 Pi, 并从数据流窗口中读取相应长度的子串;
    调用 Bloom[ i-1 ] 对所读取的子串进行匹配检验, 如匹配成功则将处理器标记位置为 1;
    /* 标记位为 1 代表该子串是可疑子串 */
  End For
  标记位为 1 的处理器将其所处理的子串提交分析器做进一步判断处理;
End

```

其中 Bloom[L] 为并行计数式 Bloom 过滤器组数组,即采用一个数组来存储 L 个计数式 Bloom 过滤器。

2.2 哈希函数的构造

在规则集通过哈希函数映射到并行 Bloom 过滤器数组的过程中,哈希函数的选择对 Bloom 过滤器性能是否高效非常关键,本节将讨论哈希函数的构造。

为了降低 Bloom 过滤器的假阳性概率,哈希函数值必须

尽可能有效均匀地分布在哈希空间上,本文采用的 murmur 哈希具有高性能和低冲突率的特点,是一种高效的哈希函数^[13]。

由式(2)可知要使 Bloom 过滤器假阳性概率最低,哈希函数的个数 k 须满足 $k = (m/n) \ln 2$,而当哈希函数个数过多时会增加计算量,从而影响 Bloom 过滤器的性能。文献[14]对超过两个以上的哈希函数提出了一种特殊的哈希技巧,在提高 Bloom 过滤器性能的同时不影响假阳性概率,该技巧可用式(4)表示:

$$g_1(x) = h_1(x) + ih_2(x) \bmod m \quad (4)$$

其中: m 为 Bloom 过滤器的长度; $h_1(x)$ 和 $h_2(x)$ 是两个独立的哈希函数; i 的范围为 $0 \leq i \leq k-1$ 。

本文采用 murmur 哈希函数结合式(4)构造哈希函数来提高 Bloom 过滤器的性能和效率。采用构造哈希函数实现的计数式 Bloom 过滤器查询算法具体描述如下:

```

Input: 需要查询的子串 key, 计数式 Bloom 过滤器组 bloom[ m ]
Output: bloom[ m ]
Begin
  h1(key) = murmur.hash(key, 0); /* 调用 murmur 哈希函数 */
  h2(key) = murmur.hash(key, h1(key));
  For i = 0 to k-1
    d = Math.abs((h1(key) + i * h2(key)) % m)
    /* 应用式(4)产生 k 个哈希值, m 为 Bloom 过滤器的长度 */
    bloom[ d ] = bloom[ d ] + 1;
  End For
End

```

2.3 计数式 Bloom 过滤器参数的设置

当一组没有重复的规则集映射存储到计数 Bloom 过滤器中时,计数器单元的值为 16(即每个计数器占 4 位)就可以基本保证计数器不会因为累加而溢出^[12],具体证明如下:

用 $P(\text{value}(c) \geq i)$ 表示第 c 个计数器的值大于或等于 i 的概率,那么

$$P(\text{value}(c) \geq i) = m \binom{nk}{i} \frac{1}{m^{nk}} \leq m \binom{nk}{i} \frac{1}{m^i} \leq m \frac{(nk)^i}{i! m^i} \approx m \frac{(nk)^i}{\sqrt{2\pi i} \left(\frac{e}{i}\right)^i m^i} \leq m \left(\frac{enk}{im}\right)^i \frac{1}{\sqrt{2\pi}}$$

由式(2)可知 $1 \leq k \leq (m/n) \ln 2$,所以

$$P(\text{value}(c) \geq i) \leq \frac{m}{\sqrt{2\pi}} \left(\frac{e \ln 2}{i}\right)^i$$

若每个计数器分配 4 bit,当计数器里的值达到 16 时溢出的概率为:

$$P(\text{value}(c) \geq 16) \leq 0.56 \times 10^{-15}$$

此值足够小,由此可以说明计数器分配 4 位对大多数的程序都能保证不会溢出。

3 实验结果及分析

为了测试性能,实验环境选用 Dell Poweredge r710 服务器作为硬件平台,该服务器拥有 2 颗 Intel E5606 2.13 GHz 的处理器,每个处理器内嵌四个核,8 MB 的三级缓存和 8 GB DDR3 SDRAM,操作系统为 Windows Server 2003,在 VS 2008 中使用 VC++ 和 OpenMP 编程实现该算法。其中计数式 Bloom 过滤器计数器为 4 bit, $L = 8$, $m = 2048$, $k = 2$, $n = (m/2) \ln 2 < 700$ 。

DPI 技术的高性能需求主要体现在存储空间开销和匹配

吞吐量等两方面,即提高其特征匹配算法速率,减少其存储空间开销。因而本文将从算法的匹配速率和存储空间这两个方面,通过理论和实验来证明该算法的时空高效性。

3.1 匹配速率

当在并行计数式 Bloom 过滤器组中更新一条规则时,该规则将插入到相应的计数式 Bloom 过滤器中,需要进行 k 次哈希运算,其插入操作的时间复杂度为 $O(k)$ 。当 L 个子字符串在计数式 Bloom 过滤器组中完成匹配检测时,由于并行性其时间复杂度相当于一个计数式 Bloom 过滤器的时间复杂度 $O(k)$,而 k 是常数,所以该算法匹配吞吐量高。

本文采用 Snort 版本 2.8.3.2, Winpcap 版本 4.0.2, 实验数据集采用本机 Snort 系统下采集的本地日志数据,这样做的目的是减少网络抓包时间的影响,减小误差,提高测试结果的准确性。

在一局域网的出口处捕获 100 MB 大小的日志数据,在同一计算机上分别用基于本文并行 Bloom 过滤器组算法、BM (Blyer-Moore) 算法、AC (Aho-Corasick) 算法的 Snort 系统运行,测试算法的匹配速率及不同 Snort 规则集大小对总的搜索时间的影响。结果如图 5 所示。

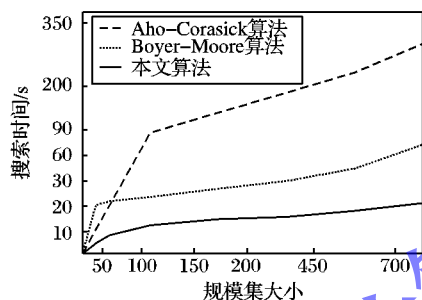


图5 算法匹配速率对比

从图 5 中比较分析可知基于并行 Bloom 过滤器算法的匹配速率大大高于基于 BM 算法或 AC 算法,同时 Snort 规则集大小对匹配速率的影响不明显。

3.2 存储空间

计数式 Bloom 过滤器组并行算法中采用 L 个长度为 $4m$ bit 的计数式 Bloom 过滤器来表示 n 条规则,其空间复杂度为 $O(4mL)$,即平均每条规则只需占用 $(4mL)/n$ bit,且存储空间与规则本身的大小无关,因而是一种存储高效的算法。

测试了算法在不同 Snort 规则集大小下所需存储空间的变化,并与经典的 BM 算法和 AC 在进行了比较,得出如图 6 所示的关系。

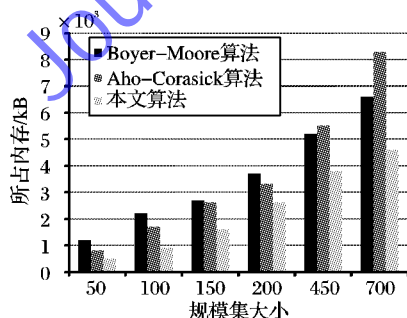


图6 存储空间的比较

从图 6 可知,基于并行 Bloom 过滤器算法的存储空间开销较小且明显低于基于 BM 算法或 AC 算法的空间需求。同时可知 AC 算法在规则集较大时其空间需求会出现膨胀性的增长。

4 结语

DPI 面临高性能的挑战,即如何满足高速数据包线速处理需求以及日益增长的存储空间需求。本文提出了一种在多核平台上基于并行 Bloom 过滤器组的深度数据包检测算法,利用多核平台并行计算能力提高性能,构造高效哈希函数降低假阳性误判概率,采用计数式 Bloom 过滤器解决规则集动态删除问题,并进行了大量实验,实验结果证明该算法是一种时空高效的算法。同时还与经典的 BM 和 AC 算法进行了时间和空间比较,充分说明该模型的实用性与高效性。

随着多核处理器等硬件技术的不断发展,如何细粒度并行化规则匹配算法将是并行数据包检测技术研究重点和难点^[3],因此下一步工作将研究如何在多个并行 CPU 核上实现规则匹配的数据并行、任务并行以及流水线等多粒度并行,从而提高深度包检测的匹配吞吐量。

参考文献:

- [1] SINGH S, ESTAN C, VARGHESE G, *et al.* Automated worm fingerprinting[C]// OSDI 2004: Proceedings of the 6th Conference on Symposium on Pearting Systems Desig & Implementation. Berkeley: USENIX Association, 2004: 4-4.
- [2] 丁晶, 陈晓岚, 吴萍. 基于正则表达式的深度数据包检测算法[J]. 计算机应用, 2007, 27(9): 2184-2193.
- [3] 黄昆, 谢高岗. 深度数据包检测技术的研究进展[J]. 信息技术快报, 2010, 8(6): 1-18.
- [4] KNUTH D E, MORRIS J H, PRATT V R. Fast pattern matching in strings[J]. SIAM Journal on Computing, 1997, 6(2): 323-350.
- [5] BOYER R S, JMOORE J S. A fast string searching algorithm[J]. Communications of the ACM, 1977, 20(10): 762-772.
- [6] AHO A V, CORASICK M J. Efficient string matching: an aid to bibliographic search[J]. Communications of the ACM, 1975, 18(6): 333-330.
- [7] 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.
- [8] SPITZNAGEL E, TAYLOR D, TURNER J. Packet classification using extended TCAMs[C]// ICNP 2003: Proceedings of IEEE International Congerence on Network Protocols. Piscataway: IEEE, 2003: 120-131.
- [9] LIU A X, MEINERS C R, ZHOU Y. All-match based complete redundancy removal for packet classifiers in TCAMs[C]// INFOCOM 2008: Proceedings of 27th IEEE International Conference on Computer and Communications. Piscataway: IEEE, 2008: 111-115.
- [10] DHARMAPURIKAR S, KRISHNAMURTHY P, SPROULL T, *et al.* Deep packet inspection using parallel Bloom filters[C]// HOTI 2003: Proceedings of the 11th Symposium on High Performance Interconnects. Piscataway: IEEE, 2003: 44-51.
- [11] BLOOM B H. Space/Time trade-offs in hash coding with allowable errors [J]. Communicatons of the ACM, 1970, 13(7): 422-426.
- [12] FAN L, CAO P, ALMEIDA J, *et al.* Summary cache: a scalable wide-area Web cache sharing protocol[J]. IEEE/ACM Transactions on Networking, 2000, 8(3): 281-293.
- [13] APPLEBY A. MurmurHash[EB/OL]. [2012-04-26] <http://en.wikipedia.org/wiki/MurmurHash>.
- [14] KIRSCH A, MITZENMACHER M. Less hashing, same performance: Building a better Bloom filter[J]. Random Struct & Algorithms, 2008, 33(2): 187-218.