高速网络超连接主机检测中的流抽样算法研究

王洪波, 程时端, 林 宇
(北京邮电大学网络与交换技术国家重点实验室, 北京 100876)

摘 要: 检测超连接主机是网络安全中的重要问题. 而流抽样是高速网络环境下解决该问题的基础. 现有解决方案使用基于哈希流抽样算法, 其基本假设是存在均匀随机哈希函数. 但是已有研究并没有评价此假设的合理性. 该文通过技术分析和实验测试得出结论: 在 2. 5Gbps 以上高速网络中, 以上假设在线性流 ID 序列情况下并不合理. 随后, 该文基于 Bloom filter 数据结构提出一种新的流抽样算法. 算法分析表明: 新算法具有 10Gbps 线速处理能力和较小的空间复杂度. 最后, 该文基于实际互联网数据进行实验评价, 结果显示: 新算法能够实现独立于流 ID 的等概率随机抽样.

关键词: 网络安全; 超连接主机; 流抽样; 哈希函数; Bloom filter
中图分类号: TN918 文献标识码: A 文章编号: 0372-2112 (2008) 04-0809-10

On Flow Sampling for Identifying Super-Connection Hosts in High Speed Networks

WANG Hong-bo, CHENG Shi-duan, LIN Yu
(*State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing* 100876, *China*)

Abstract: Detecting super-connection hosts is an important issue in network security and flow sampling is the key to solve this problem in high speed networks. The existing solutions use hash-based flow sampling algorithm, which assumes that the uniform random hash functions are available. However, this assumption can not be justified. By technical analysis and experiment tests, this paper concludes that the assumption is not true for linear flow IDs in high speed networks (above 2. 5Gbps). A new flow sampling algorithm is presented subsequently, which exploits the Bloom Filter data structure. An analysis demonstrates that the new algorithm can support the 10Gbps line-speed processing with low space complexity. Experiments are also conducted based on real network traces. Results show that the proposed algorithm can achieve equal probability flow sampling independent of flow ID distribution.

Key words: network security; super-connection host; flow sampling; hash functions; Bloom filter

1 引言

近年来网络入侵事件频繁发生, 如分布式拒绝服务攻击(DDoS)、蠕虫(Worms)传播、端口扫描(Port Scan)等, 有的甚至造成了巨大的经济损失. 随着互联网逐渐渗透到人类社会各个方面, 设计准确、快速的网络入侵检测方法已经成为网络研究中的重要课题.

经研究发现, 网络入侵所具有的相似行为有利于检测的进行. 例如, 扫描式蠕虫(Scanning Worms)在进行传播时, 其被感染主机通常在短时间内向大量其他主机发送分组. 这反映在网络上则表现为: 在短时间内存在一系列 IP 分组具有同一源 IP 地址、不同目的 IP 地址(或同一目的 IP 地址不同端口号). 与以上情况对称, DDoS 则是在短时间内大量不同主机向同一被攻击主机发送攻击分组, 在此情况下则是一系列 IP 分组具有同一目的 IP 地址、不同源 IP 地址. 上述所有情况的共同特点是: 在短时间内某主机与其他主机间存在大量不同的 IP 流(IP flow), 我们称此种情况下的主机为超连接主机. 本文中所述的 IP 流是一系列具有共同属性的 IP 分组集合, 其中的共同属性由流 ID(flow ID)决定. 流 ID 由两部分组成: 源部、目的部, 其中"源部"是分组头部中的源 IP 地址(或加上源端口号), "目的部"是分组头部中目的 IP 地址(或加上目的端口号). 在以上例子中, 蠕虫传播主机以及 DDoS 的被攻击者都是超连接主机. 超连接主机的存在预示可能有网络入侵事件发生, 并可以此为触发条件采取进一步的分析和响应. 因此, 检测超

收稿日期: 2007-05-17; 修回日期: 2008-01-31
基金项目: 国家自然科学基金(No. 90604019, 60502037); 国家 973 重点基础研究发展规划(No. 2003CB314806); 国家 863 高技术研究发展计划(No. 2006AA01Z235)

连接主机是网络入侵检测中的一个重要问题.

目前, 网络入侵检测系统已广泛部署于子网的边缘或网络的入口处以保护企业的内部网络. 近年来, DDoS 在攻击目标的同时, 大量消耗网络带宽; 蠕虫的传播往往使相关链路迅速达到饱和. 这些入侵事件都会导致互联网服务提供商(ISP) 的核心网络服务质量下降, 促使越来越多的 ISP 正在或计划在其核心网络部署网络入侵检测系统. 另外, 大型企业内部网络及其 Web 站点往往是 DDoS 的攻击目标, 而随着业务的发展它们的网络接入带宽也不断得到提升. 这就需要网络入侵检测能够适应更高的链路速率(2.5Gbps 以上) 以满足 ISP 或大型企业的需要.

传统的超连接主机检测方案[1~4] 大都使用维护每流状态的方法. 例如, 为了检测端口扫描, Snort 系统[2] 通过保存每个流信息(即由源、目的对组成的流 ID) 以统计每个源主机与多少不同主机(或端口) 进行通信. 然而此方法不能扩展到高速网络环境中. 一方面, 在高速链路下, 单位时间内存在大量的流(Flow), 为了保存大量的流 ID, 必须使用高容量的存储器; 另一方面, 为了在高速网络中线速处理 IP 分组, 必须使用高速存储器. 根据现有的半导体技术, 动态随机访问存储器(DRAM) 容量大, 但是其访问速度较慢(几十纳秒) 不能适应高速网络中的线速处理需求; 而静态随机访问存储器(SRAM) 访问速度虽快(几纳秒), 但是其容量有限(几十 Mbits), 不能满足维护每流状态的要求.

为了应对这种网络高速化所带来的可扩展性挑战[5,6], 最近的研究[7~9] 提出采用基于流抽样(Flow Sampling) 的方法. 其基本思想是对每个新观测到的流进行固定概率的抽样, 与抽样流相关的主机(源 IP 地址主机或目的 IP 地址主机) 被列为重点测量对象并只对它们进行流测量, 在以后的测量中如果某个重点测量对象的流个数超过一定的门限, 则将被判断为超连接主机. 因为与超连接主机相关的流的数量较多, 只要流抽样概率设得合适超连接主机将在概率保证下被列为重点测量对象.

由于在测量点进行测量的基本单位是 IP 分组而流抽样的单位是流, 并且一个流可能包括多个分组, 因此流抽样是一个具有挑战性的问题. 现有研究[7~9] 都使用基于哈希流抽样(hashr-based flow sampling) 方法, 其基本假设是存在哈希函数对流 ID 的映射具有充分的均匀性. 但是这些研究并没有讨论这个假设条件的合理性. 在高速网络环境下假设条件是否对于所有的可能情况都成立呢? 若不成立, 是否有其他的流抽样算法可供选择? 这就是本文的研究动机.

本文的贡献主要体现在三个方面:

(1) 总结了流抽样的本质需求和哈希函数的选择因素; 以此为基础对基于哈希流抽样算法的基本假设进行了考察, 并得出结论: 在高速网络环境下假设条件并不完全成立.

(2) 提出了一个新的基于 Bloom Filter 数据结构的流抽样算法以用于高速网络环境.

(3) 基于实际互联网数据, 对新算法进行了验证并与基于哈希抽样算法进行了比较; 给出了两个算法的适用范围, 为系统设计和实现者提供了选择依据.

本文后续部分组织如下: 下一节介绍相关工作. 第 3 节对基于哈希流抽样的基本假设进行了分析和实验评价. 第 4 节详细描述我们的新算法. 第 5 节进行实验验证. 最后一节总结全文.

2 相关工作

直接且简单的超连接主机检测方法是维护每流状态. 例如: Bro[1]、Snort[2]、FlowScan[3]、CoralReef[4] 等都使用哈希表保存所有流 ID 以发现超连接主机. 如前所述, 这种方法由于缺乏可扩展性而只能使用在低速网络环境下.

近年来, 文[10, 11] 分别引入了无状态的可扩展检测算法, 但是它们仅支持 DDoS 检测, 很难应用到其他类型的具有超连接主机的入侵检测中. 为了在高速网络上检测更广泛的入侵类型, Levchenko 等人[5] 率先在理论上对入侵检测的可扩展性进行了研究并探讨了该问题的困难性. 在此基础上, 文[6] 通过考虑攻击的具体语义, 设计了"部分完成过滤器"(Partial Completion Filter) 及可扩展算法以检测几类特殊的攻击, 在可扩展性检测算法方向做了有益的起始研究.

从算法设计角度看, 高速网络中的超连接主机检测算法必须同时具有较低的时间和空间复杂度. 一方面, 为了满足 IP 分组的线速处理需求, 算法必须足够简单以使得每个分组的处理时间小于等于链路上分组到达的最小时间间隔; 另一方面, 算法应尽可能少地保存状态信息, 以便使用容量有限、价格昂贵的高速存储器(如 SRAM). 具有简单和高速优点的随机算法[12] 满足以上要求. 因此, 最近的研究[7~9] 采用随机化思想来解决检测超连接主机的可扩展问题, 并以流抽样为基础. 文[7] 首次指出了超连接主机问题并提出通过流抽样来提高可扩展性, 文中给出了一级过滤(One level Filtering) 及二级过滤算法(Two Level Filtering), 它们都使用基于哈希流抽样方法来达到流抽样目的. 随后的研究[8,9] 主要是在此文基础上集中在对流计数的改进. 文[8] 针对"flow hog" 提出与文[7] 中一级过滤算法思想类似的"flow sampling and hold" 算法. 为了进行近似流计数, 该文提出了基于 Bloom Filter、List-Triggered Bitmaps 两个流计数算法. 最近, 文[9] 把流抽样与数据流(data streaming) 技术

相结合提出了两个新算法以适应更高的速率需求. 该文第一个算法在流抽样后引入一种简单的数据结构(实际是一维的 Bloom Filter) 以减少对哈希表的访问次数. 该文第二个算法引入一种二维比特矩阵数据结构作为流近似计数的基础, 同时把流计数与流抽样功能进行分离以使两个功能模块可以并行运行. 以上研究中的流抽样都毫无例外地使用基于哈希流抽样算法, 并假设存在哈希函数能够为流 ID 提供足够均匀的随机映射, 但是都没有深入讨论其假设的合理性. 本文弥补这方面的不足, 并提出新的流抽样算法. 有必要指出: 本文使用 Bloom Filter 用于流抽样, 并非用于流计数[8,9].

3　基于哈希流抽样算法及其缺陷

对于超连接主机检测来说, 流抽样要满足两个基本要求: (1) 抽样的一次性: 在网络层, 测量的基本单位是分组而一个流会有多个分组, "抽样的一次性" 要求: 对一个流进行抽样的机会只有一次, 而不依赖于实际测量到属于该流的分组数量, 这就保证了抽样的基本单位是流而不是分组; (2) 等概率随机抽样: 任何一个流被抽样的概率是相等的而不依赖于流 ID 的内容, 这就保证了一个主机被列为重点测量对象的概率与它的流数量成正比, 从而使得超连接主机在概率保证下被列为重点测量对象.

本节后面部分首先介绍基于哈希流抽样算法, 然后通过分析及实验指出其存在的缺陷.

3.1　基于哈希流抽样(hashr based flow sampling)

现有文献[7~9]采用的流抽样算法是基于哈希流抽样, 其基本思想是: 给定一个哈希函数 h, 其定义域为流 ID 空间 F, 值域为 $H:[0,1)$; 再给定一个抽样域 $S:[x,x+r)$, 其中 $0\leqslant x<x+r\leqslant 1$, 称 r 为抽样概率; 对于每个到达的分组, 记它所属流的流 ID 为 f, 求它的哈希值 $h(f)$, 如果 $h(f)\in S$ 则该分组所在的流被抽样, 否则不被抽样.

由于属于同一个流的所有分组都有同样的流 ID f 而哈希函数 h 是确定性函数, 所以在测量到每个属于同一个流的分组时所计算的哈希值 $h(f)$ 是相同的, 从而满足了"抽样的一次性"要求.

另一方面, 给定哈希函数 h 以及抽样域 $S:[x,x+r)$, 一个流是否被抽样是依赖于流 ID f 的(即是否 $h(f)\in S$). 为了满足"等概率随机抽样"的要求, 哈希函数 h 必须满足均匀随机哈希(uniform random hashing)属性: 对于任意 f, 哈希值 $h(f)$ 是均匀分布于 $H:[0,1)$ 上的随机变量. 在理论上并不能证明存在哈希函数满足均匀随机哈希属性[13], 因此使用基于哈希流抽样的基本假设前提是存在哈希函数有足够近似的均匀随机哈希属性.

事实上, 在网络测量领域使用哈希函数进行随机抽样的思想最早出现于文[14], 该文通过在不同测量点使用相同哈希函数对 IP 分组进行抽样以保证不同测量点具有相同的分组样本集合, 从而解决了分布测量环境下分组抽样的一致性问题. 虽然基于哈希的方法可以应用于分组抽样, 但并不意味着一定能很好地应用于流抽样中.

3.2　哈希函数的需求及其选择

由上节可知, 基于哈希流抽样算法的关键是哈希函数的选择. 在检测超连接主机应用中, 基于哈希流抽样算法需要哈希函数满足以下三个要求: (1) 能够快速计算; (2) 安全性高; (3) 具有足够好的均匀随机哈希属性.

由于要对链路到达的每个 IP 分组进行线速处理, 因此需要哈希函数能在最短的分组到达时间间隔内* 完成一次哈希值的计算. 例如: 在 OC-48(2.5Gbps)、OC-192(10Gbps) 链路下每哈希值的计算应分别小于 128 纳秒、32 纳秒. 这就要求哈希函数应具有很快的计算速度以适应不同的高速链路.

超连接主机检测主要应用于网络安全领域, 而检测系统本身的安全性更为重要. 由于基于哈希流抽样使用哈希函数进行流抽样, 因此它易受到文[15]所描述的算法复杂性攻击而存在安全隐患. 例如, 在 DDoS 攻击中, 如果事先掌握了流抽样中使用的哈希函数, 攻击者就可以通过伪造源 IP 地址以产生一系列特定的攻击流以使得这些流 ID 的哈希值都落在(或大部分落在) 抽样域之外从而逃避超连接主机检测. 根据文[15]建议, 较好的解决方法是使用带初始参数的哈希函数(如 MD5, SHA1, CRC32 等), 每次运行时随机选择初始参数从而保证哈希函数不可预测. 更好的解决方法是每次从 Universal 哈希函数族[16]中随机选取哈希函数.

虽然能够用哈希函数提供伪随机哈希值序列, 但是由于哈希函数是确定性函数, 哈希值仍然依赖于哈希函数的输入值. 如果输入具有随机性, 则比较容易得到较好的随机性输出; 如果输入没有随机性而又要得到足够随机的输出就必须使用具有强均匀随机哈希属性的哈希函数. 相对于分组抽样, 流抽样对哈希函数的均匀随机哈希属性要求更高. 一方面, 在分组抽样中可以通过增加哈希函数的输入长度来提高输入序列的随机性来提高抽样的随机性[14]. 但是在流抽样中, 流 ID 的组成是固定的, 只能由有限的域组成(如: 源、目的 IP 地址, 源、目的端口号). 另一方面, 分组抽样中还可选取具有较高随机性的字段来增加输入的随机性[14], 而由于流 ID 的固定性这种方法是不可行的; 更具挑战性的是流抽样要求哈希函数应该把同一超连接主机所有流的流 ID 均匀映射到哈希值域上, 但是属于同一超连接主机

* 网络系统设计中, 最短 IP 分组到达时间间隔往往取最小 IP 分组长度与链路速率的商, 其中最小 IP 分组长度常取 40 字节.

的所有流 ID 都至少具有相同的源部或目的部,而且在很多情况下流 ID 序列是线性变化的(如:顺序扫描式蠕虫产生的目的 IP 地址是连续的,端口扫描中产生线性目的端口号以寻找具有潜在漏洞的端口号等等),我们称这种流 ID 序列为线性流 ID 序列. 对于其他无特定规律的流 ID 序列我们通称为随机流 ID 序列. 图 1 是一个线性流 ID 序列的例子,其中的流 ID 由源、目的 IP 地址组成,各流 ID 中的目的 IP 地址依次递增. 线性流 ID 序列的特点是各个流 ID 间只有较少的比特存在差异,其他比特都相同. 它们作为哈希函数的输入具有较弱的随机性. 所以基于哈希流抽样方法所使用的哈希函数也应该能够对线性流 ID 序列产生足够均匀的随机分布. 由此可见,基于哈希流抽样与基于哈希分组抽样存在很大的不同. 虽然基于哈希的方法可以应用于分组抽样,但并不一定能很好地应用于流抽样中. 文[14]在引入基于哈希分组抽样时对哈希函数进行了实验评价,而目前的研究在使用基于哈希流抽样时并没有注意到流抽样与分组抽样的差异并对可以使用的哈希函数进行进一步的考察.

⟨a. b. c. d, e. f. g. h⟩, ⟨a. b. c. d, e. f. g. h+ 1⟩,
⟨a. b. c. d, e. f. g. h+ 2⟩, ⟨a. b. c. d, e. f. g. h+ 3⟩ …

图 1 线性流 ID 序列举例

目前公认能产生随机性序列最好的哈希函数是 MD5、SHA1 等加密性哈希函数,但是这些哈希函数计算复杂很难做到快速计算,即使用专用的硬件实现也需要几十个时钟周期来产生一次输出,因此很难应用到高速的分组处理环境中. 例如,一些现有的硬件实现[17]至少需要 64 个时钟周期,这已经超过了高速网络环境下最小的分组处理时间*. 而且,由于 MD5、SHA1 等算法的顺序操作本质很难再通过进一步的并行及流水优化技术来提高计算速度. 同样,CRC32 算法由于具有大量乘除法运算,其计算速度也较慢. 而基于简单除法、乘法运算的哈希函数[13]由于其数学结构简单而缺乏安全性.

因此,现有研究[8,9]在使用基于哈希流抽样时都倾向于使用 H_3 哈希函数族[16]. 由于 H_3 哈希函数的运算仅由简单的"与"和"异或"逻辑组成,因此它很容易由硬件实现[18]且计算速度能达到纳秒级. 又由于每次程序运行时可以从 H_3 哈希函数族中随机选择某个 H_3 函数,因此攻击者很难预测哈希函数从而使用此哈希函数具有很强的安全性.

虽然 H_3 哈希函数族具有计算快速和安全的优点,但是目前并没有研究验证 H_3 哈希函数是否对流 ID 序列(特别是线性流 ID 序列)产生足够好的均匀随机输出以适用于流抽样.

3.3 卡方检验及算法缺陷

为了验证当前可用于流抽样的哈希函数是否有足够的均匀哈希属性,我们使用数理统计中的卡方(chi-square)检验方法对哈希函数生成哈希值的分布进行均匀性检验.

我们把哈希函数的值域 $[0, 1)$ 等分为 k 个小区间(bin),用哈希函数 h 计算 n 个流 ID 的哈希值,记落在第 i 个小区间上的哈希值个数为 Y_i $(i= 1, 2, \cdots, k)$;如果哈希函数 h 满足均匀随机哈希属性,则落在每个小区间上哈希值个数的期望值应为 $y= n/k$;使用卡方统计量:

$$x^2= \sum_{i=1}^{k}(Y_i- y)^2/y \qquad (1)$$

进行假设检验:假设由哈希函数 h 计算的哈希值服从均匀分布,则统计量 x^2 是具有 $k-1$ 个自由度服从卡方分布的随机变量;给定一个显著水平 α(本文取 $\alpha= 0.01$)可通过卡方分布计算第 $1-\alpha$ 百分位数 $x_{1-\alpha}$,当统计值 $x^2> x_{1-\alpha}$ 时,我们以 $1-\alpha$(在本文中为 99%)的信任水平拒绝原假设,即拒绝哈希值服从均匀分布.

实验中我们取三种哈希函数:H_3、Bob、CRC32. H_3 被认为是最适合进行流抽样的哈希函数;Bob 是 PSAMP 工作组推荐用于分组抽样的哈希函数;CRC32 计算速度慢不适合用于高速网络环境中,但是它具有较强的随机哈希属性,我们仍然对其进行实验以作为对比.

实验中,我们使用两种类型的流 ID 序列:线性流 ID 序列、随机流 ID 序列. 每个序列中的流 ID 都具有相同的"源部"或"目的部",也就是说每个序列都是跟某个超连接主机有关的流 ID. 我们取不同的 k 值:10、100、200、400、600、800、1000,它们分别对应取不同的抽样率 $1/k$;对每个 k 值,流 ID 序列的流 ID 个数 n 取 $n= ck$,其中 $c= 5, 6, \cdots, 12$;在不同的 k、n 取值下,我们对于每种流 ID 序列随机产生 1000 个具有 n 个流 ID 的流 ID 序列,然后对每个流 ID 序列使用以上哈希函数计算哈希值并进行卡方检验. 对于每个流 ID 序列,Bob、CRC32 哈希函数只计算一次,而对于 H_3 哈希函数,则是随机选取 100 个 H_3 哈希函数分别进行计算. 最后我们对不同 k、n 值情况下,统计假设检验的结果.

图 2 是对线性流 ID 序列的结果统计($c= 10$, c 取其他值时结果相同). 可以看到:对于不同的 k 值(对应于不同的抽样率 $1/k$),H_3 哈希函数都有 17%~20% 的情况被卡方检验拒绝具有均匀随机哈希属性,Bob 哈希函数的情况更差:在 40%~85% 之间;CRC32 在 k 取 10、100、200 时,均匀性最好,而当 k 大于 400 时 90% 以上的情况不能通过均匀性检验. 而对于随机流 ID 序列,基本

* 例如:即使用 Xilinx FPGA Virtex 系列当前最高档产品 Virtex-5 (550MHz),64 个时钟周期也至少需要 116 纳秒,而在 10Gbps 链路下,最小分组的处理时间应小于 32 纳秒.

都能通过均匀性检验(本文不再列出其统计结果图).

为进一步分析均匀随机哈希假设对超连接主机检测结果的影响,我们考察了不能通过均匀性检验时哈希值分布情况.作为对比,图 3 是当哈希值均匀分布的情况($k=10$, $n=100$).可以看出,无论抽样域在哪个区域流 ID 的抽样个数都将接近于期望值.而图 4 则是哈希值分布不均匀的情况($k=10$, $n=100$),如果抽样域选择在[0.4, 0.5)或[0.5, 0.6),虽然同一个超连接主机有 100 个流,也因为抽不到样本或抽到的样本太少从而遗漏掉该超连接主机;同样,图 5 是当时的哈希值分布不均匀的情况.

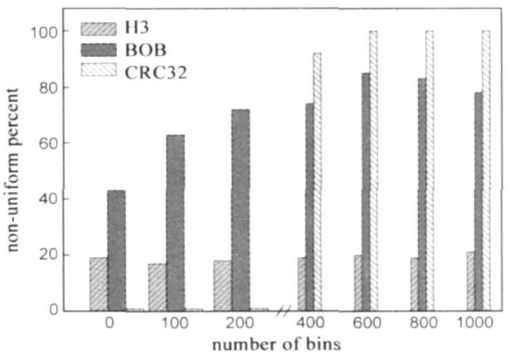

图 2　卡方检验统计结果

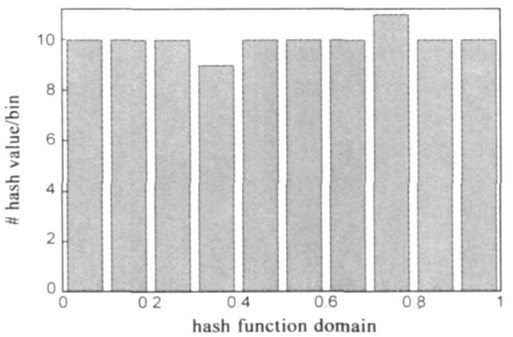

图 3　哈希值均匀分布情况 (n=100)

由以上可知:现有可用于高速网络环境下的哈希函数很难对线性流 ID 序列产生足够好的均匀随机输出;而线性流 ID 序列在蠕虫传播、端口扫描等入侵事件中是一种重要的流 ID 序列,不能忽略.因此,基于哈希流抽样方法对于均匀随机哈希的假设在高速网络环境下并不完全成立,这种方法只能使用在速率较低的环境下(低于 2.5Gbps).为了能在更高速的环境下检测超连接主机,有必要设计新的流抽样算法.

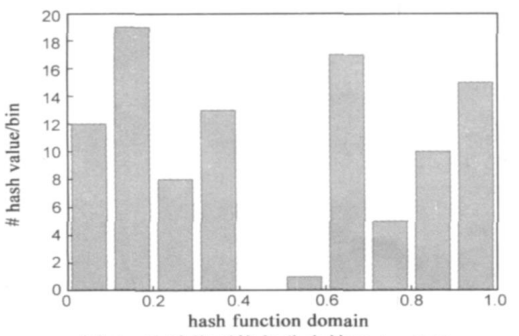

图 4　哈希值不均匀分布情况 (n=100)

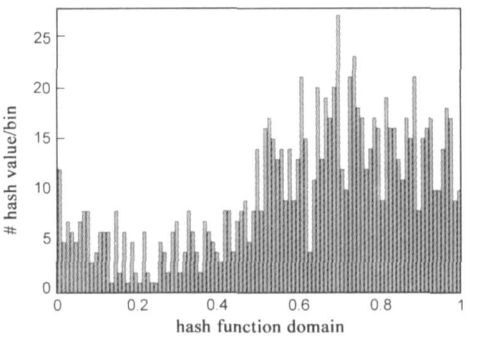

图 5　哈希值不均匀分布情况 (n=1000)

4　基于 Bloom Filter 流抽样算法

基于哈希流抽样算法试图通过哈希函数来同时解决流抽样的两个基本需求:“抽样的一次性”和“等概率随机抽样”,但是由于对于哈希函数的均匀随机哈希属性假设不合理导致这种方法不能扩展到高速网络环境中.事实上,基于哈希的抽样方法是通过哈希函数的“确定性”来解决“抽样的一次性”,用哈希函数一定的“随机性”来处理“等概率随机抽样”.虽然哈希函数的随机性不能解决好“等概率随机抽样”,但是这并不能否定哈希函数对解决“抽样的一次性”作用.

因此我们设计流抽样算法的基本思想是:把“等概率随机抽样”从哈希函数提供的功能中分离出来而由另外的模块处理,从而减弱对哈希函数的随机性要求;同时,仍由哈希函数提供“抽样的一次性”.问题是如何设计另外的模块来对流提供等概率随机抽样功能.当一个新流的第一个分组到达时如果我们能判断出它属于一个新流,问题就可迎刃而解,这时我们只需产生一个随机数* 来决定是否抽样该流.关键是如何提供这样一个机制来判断一个新流的到达.最简单的方法是维护每流状态:提供一个哈希表;每个分组到达时通过流 ID 查找该表,若表中没有该流 ID 则说明是一个新流到达,把该流插入到哈希表中;若表中有该流 ID 则说明该分组是属于以前的流,没有新流到达.但是这种方法要使用大量的高速存储器(每个流 ID 至少需要 8 字节)而在高速网络环境中不可行.这就需要空间高效的数据结构来提供这种集查询功能.一种被称为“Bloom Filter”的数据结构满足了这样的要求.

4.1　Bloom filter 简介

* ③伪随机数生成器的硬件实现已经广泛存在于各种网络设备中,而且能提供理想的随机性.

Bloom filter是一种简单而且空间高效的数据结构,常用它来表示一个集合以提供成员查询操作. 此数据结构最早由 Burton Bloom[19] 于 20 世纪 70 年代引入, 近年来在网络领域中得到广泛的应用[20].

记一个具有 n 个元素的集合 $X=\{x_1, x_2, \cdots, x_n\}$, 表示 X 的 Bloom filter 可以用一个 m 比特的位向量 v 以及 k 个独立的哈希函数 $h_1, h_2, \cdots, h_k$ 来实现, 其中每个哈希函数的值域都为 $\{1, 2, \cdots, m\}$. 开始时, v 中所有比特初始化为 0; 对于每个元素 $x \in X$, 置 v 中的第 $h_1(x)$, $h_2(x)$, $\cdots$, $h_k(x)$ 比特为 1(有可能一个比特位已经为 1); 当查询一个元素 y 是否属于集合 X 时, 检测 v 中的第 $h_1(y)$, $h_2(y)$, $\cdots$, $h_k(y)$ 比特位, 如果这 k 个比特位中有任何一个为 0, 则 y 肯定不属于集合 X; 如果 k 个比特位均为 1, 则以一定的误差概率认为 y 属于集合 X. 引起误差的原因是: 一个本来不属于集合 X 的 y, 对于每个哈希值 $h_i(y)$ ($i=1, 2, \cdots, k$) 恰好都存在一个 $x_t \in X$ ($t=1, 2, \cdots, n$) 的某个哈希值 $h_j(x_t)$ ($j=1, 2, \cdots, k$) 与之相等, 即 $\forall i \exists j \exists t(h_j(x_t)=h_i(y))$, 从而使得 v 中的第 $h_1(y)$, $h_2(y)$, $\cdots$, $h_k(y)$ 比特位由于哈希冲突而被其他属于集合 X 的元素置为 1. 假设所有哈希函数具有均匀随机哈希属性, 则 Bloom filter 的误差概率为:

$$p=\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^k \approx (1-e^{-kn/m})^k \qquad (2)$$

由上式可知, 当 n 不变时, 可以增大 m 或 k 以降低误差概率 p, 使之控制在应用可以接受的范围内.

虽然出于数学推导需要, 式(2)的假设前提是哈希函数满足均匀随机哈希属性, 但是已有研究[21]表明: 使用 Universal 哈希函数时, Bloom filter 的实际误差概率仍能接近通过式(2)计算的理论值.

4.2 算法描述

如图 6 所示, 新的流抽样算法由 Bloom filter、误差吸收(Absorbing error)及随机抽样(Random sampling)三个模块组成. Bloom filter 模块的功能是判断是否有一个新流到达; 随机抽样模块的功能是对新流进行随机抽样, 这里的抽样与每个新流的流 ID 无关; 由于 Bloom filter 存在误差(即有些新流不能被它判断出), 误差吸收模块的功能是根据误差概率来调整抽样率从而把 Bloom filter 中误差所引起的随机性"吸收"到抽样的随机性中, 从而满足"等概率随机抽样"的需求. 其中的 flow counter 用以对插入到 Bloom filter 中的流进行计数.

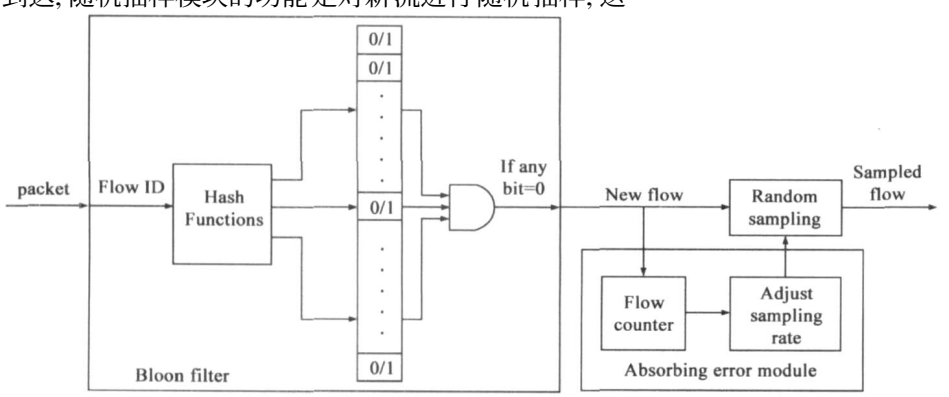

图 6 基于 Bloom filter流抽样算法模块图

整个算法的伪代码见图 7. 当一个分组到达时, 首先计算其流 ID 的 k 个哈希函数值(在图 6 中 k 取3), 若位向量 v 中的 k 个相应位均为 1, 则认为没有新流到达, 算法结束并继续处理下一个分组; 否则, 说明有新流到达, 先把该流插入到 Bloom filter 中(即把 k 个相应位置 1), 然后根据此分组到达前 Bloom filter 中已插入的流个数按照式(2)计算误差概率 p(第 7 行); 由于 Bloom filter 存在误差概率 p, 也就是说一个新流被 Bloom filter 成功检测出的概率为 $1-p$, 当 Bloom filter 发现一个新流时, 如果调整随机抽样模块的抽样概率为 $\frac{r}{1-p}$(r 为整个算法的抽样概率), 则可以保证任何一个新流被整个算法抽样的概率都等于 r(即 $(1-p)\times\frac{r}{1-p}$), 从而满足了"等概率随机抽样"需求, 以上就是误差吸收模块中的抽样率调整操作(第 8 行); 最后根据调整后的抽样概率对 Bloom filter 发现的新流进行随机抽样(第 10、11 行).

```
Bloom filter based flow sampling algorithm
1.   x= getFlowID(packet)
2.   for(i= 1 to k)
3.        if(v(h_i(x)) = = 0)
4.             v(h_i(x))= 1
5.             isNewFlow= true
6.   if(isNewFlow= = ture)
7.        p= computeErrorProbability (numOfFlows)
8.        adjustedSamplingRate= r/(1- p)
9.        numOfFlows= numOfFlows+ 1
10.       randomNumber= rand(0, 1)
11.       if(randomNumber ⇐ adjustedSamplingRate) sample this flow
```

图 7 基于 Bloom filter 流抽样算法伪代码

4.3 算法分析及实现考虑

本节首先对算法引入的空间代价进行分析, 然后从实现的角度阐明本算法可用于高速网络环境.

算法引入的空间代价主要是 Bloom filter 的位向量所占用的存储空间. 下面分析 Bloom filter 使用的存储空间. 从上节实验结果可知, 当插入到 Bloom filter 中的流个数超过一定限度后, 实际误差率将逐渐偏离于

理论误差率. 在实现时, 可以设定装载因子 β 可达到的一个上界 β_{max}(如: 当 $k=3$ 时, 取 $\beta_{max}=0.6$, 当 $k=5$ 时, 取 $\beta_{max}=0.8$), 当流计数器中记录的流个数 $n \geqslant \frac{m}{k}\beta_{max}$ 时, Bloom filter 使用另外一个空的位向量并开始新的测量周期, 而同时对暂时不用的位向量进行初始化, 两个位向量交替使用以保持测量的连续, 因此 Bloom filter 总共需两个位向量. 记每个位向量可记录最大流个数为 n_{max}, 由 $\beta=kn/m$ 每个位向量占用空间为 $m=k\frac{n_{max}}{\beta_{max}}$ (bits), 算法总共所需存储空间为:

$$u_b = 2k\frac{n_{max}}{\beta_{max}} \tag{3}$$

根据当前的半导体技术, 即使是片内(on-chip) SRAM 的容量也可以达到 10Mbits. 例如, 大部分现代商用 FPGA(现场可编程逻辑阵列)都包含多个双端口嵌入式片内 SRAM. Xilinx 公司的 Virtex-5 产品最多可包含 324 个 36kbits 双端口 SRAM 模块, 总容量可达 11Mbits[22]. 若 Bloom filter 位向量总容量 u_b 为 10Mbits, 取 $k=3$, $\beta_{max}=0.6$, 则 n_{max} 可达 1M.

为进一步衡量算法空间代价, 我们把新算法与维护每流状态方法进行比较. 记每个流 ID 长度为 l 比特, 由于维护每流状态方法需要保存每流 ID 所以维护每流状态方法所需的空间至少为 $u_f=2l\cdot n_{max}$. 定义新算法的相对空间代价为 $\rho=u_t/u_f$, 则:

$$\rho = \frac{k}{l\cdot\beta_{max}} \tag{4}$$

在 IPv4 网络中, 流 ID 长度 $l \geqslant 64$(流 ID 至少由 IP 源、目的地址组成), 当取 $k=3$, $\beta_{max}=0.6$ 时, $\rho \leqslant 7.81\%$; 若在 IPv6 网络中, 由于流 ID 长度 $l \geqslant 256$, 所以 $\rho \leqslant 1.95\%$. 即新算法引入的空间代价最多只是维护每流状态算法的 7.81%(或 1.95%). 可见新算法的相对空间代价是较低的, 尤其在 IPv6 网络中如此.

下面分析算法的处理速度. 显然, Bloom filter 中哈希函数的计算速度和位向量的访问速度决定了整个算法的最快处理速度. 由于 H_3 哈希函数仅由"与"、"异或"逻辑组成, 使用硬件实现 k 个并行的 H_3 哈希函数是容易的, 而且处理速度能达到纳秒级[23]. 而 Bloom filter 仅使用少量的存储空间, 所以可以使用高速的 SRAM. 为了提供 k 个哈希地址的并行访问, Bloom filter 位向量可以采用高速的多端口 SRAM 模块[24], 或者多个并行的双端口 SRAM. 当前, 片内 SRAM 访问速度可达 1 至 2 纳秒, 而片外(off-chip) SRAM 也可达 2 至 5 纳秒. 因此 Bloom filter 处理一个分组的时间可控制在 10 纳秒左右, 足够支持 10Gbps 链路上 IP 分组的线速处理.

综上所述, 通过引入较小的空间代价我们的新算法能够扩展到 10Gbps 高速网络环境中.

5　实验

流抽样算法是检测超连接主机的基础, 因此流抽样是否满足"等概率随机抽样"将直接影响其上层超连接主机检测应用的实际性能. 本节我们从上层应用性能的角度出发, 基于实际互联网数据进行实验对新算法和已有算法的"等概率随机抽样"属性进行评价和比较.

实验数据由背景流和异常流数据组成. 背景流数据采用互联网数据分析合作协会(CAIDA[25])及美国应用网络研究国家实验室(NLANR[26])提供的实际互联网数据, 相关信息见表 1. 我们使用不同组织在不同网络、不同时期采集的多组数据是为了保证实验数据具有较好的多样性和代表性. 异常流数据为人工生成的攻击或病毒传播数据, 它们根据实验需要加入到背景流中.

实验中, 我们采用三种不同的流定义以针对不同的应用情况. 第一种流定义的流 ID 由源 IP 地址及目的 IP 地址组成; 在检测 DDoS 攻击中常采用此种流定义, 此种情况下的异常流 ID 序列具有相同的目的 IP 地址不同的源 IP 地址. 第二种流定义的流 ID 由源 IP 地址、源端口号及目的 IP 地址组成; 它常在检测病毒传播中使用, 此种情况下的异常流 ID 序列具有相同的源 IP 地址及源端口号, 不同的目的地址. 最后一种流定义的流 ID 由源 IP 地址、目的 IP 地址及目的端口号组成; 它所对应的应用为检测端口扫描, 此种情况下的异常流 ID 序列具有相同的源、目的 IP 地址, 不同的目的端口号.

表 1　背景流数据信息

Data Source	Location	Link speed	Date
OC48 Link A (CAIDA)	One ISP in US	2.5Gbps	August 14, 2002
			April 24, 2003
			January 15, 2003
Abilene-III(NLANR)	Internet2	10Gbps	June 1, 2004
CENIC-I(NLANR)	CENIC	10Gbps	March 18, 2005

5.1　评价指标

在超连接主机检测应用中, 流抽样算法有两种使用方式: 一种是先对所有流进行抽样, 然后根据抽样结果来估计与某个主机相关的流个数, 再把流个数超过某个阈值的主机判断为超连接主机[7]. 在这种情况下, 由于使用估值会产生两种错误: 非超连接主机被误认为超连接主机, 称之为"误检"(false positive)并定义"误检率"为误检个数与非超连接主机总数的比值; 超连接主机没有被检测到, 称之为"漏检"(false negative)并定义"漏检率"为漏检个数与超连接主机总数的比值. 我们将通过超连接主机检测实验中的"误检率"与"漏检率"来评价和比较流抽样算法, 并称之为流抽样算法的"准确性"评价指标.

另一种使用流抽样算法的方式是: 先进行流抽样, 一旦某个流被抽样则与之相关的主机被列为嫌疑主机, 然后使用其他更准确的算法针对嫌疑主机进行流计数, 再把流个数超过某个阈值的嫌疑主机确定为超连接主机[8,9]. 在这种情况下, 流抽样的作用是以概率保证每个超连接主机发出(或接收)的流中至少有一个流被抽样从而使主机被列为嫌疑主机. 按照经过检测点的先后顺序, 一个超连接主机发出(或接收)的流依次记为: f_1, f_2, f_3, …, 若第一个被抽样的流为 f_{n_s}, 则 n_s 是一个随机变量. 又由于流抽样是以 r 为抽样概率的"等概率抽样", 所以 n_s 应该是服从几何分布的随机变量(参数为 r), 其均值 $E(n_s)$ 应该为 $1/r$. $E(n_s)$ 表征的是从平均意义上当超连接主机发出(或接收)多少个攻击流时能被列为嫌疑对象, 它是决定检测系统反应速度的一个重要因素. 而 $E(n_s)$ 是由流抽样算法决定的, 因此我们称之为流抽样算法的"及时性"评价指标. 可以通过判断此指标是否符合理论值(即 $1/r$)来验证流一个抽样算法的"等概率随机抽样"属性, 也可以通过比较此指标的大小来区别流抽样算法的好坏.

5.2 "准确性"评价

我们使用文[7]中的一级过滤算法检测超连接主机以比较流抽样算法. 假设流个数超过 k_s 的主机为超连接主机, 根据用户给定的参数 b ($b>1$), 如果流个数小于 k_s/b 的主机被判断为超连接主机, 则称"误检"发生. 文[7]证明: 对于任意给定的信任度 δ ($0<\delta<1$), 一级过滤算法能够自动选择合适的抽样率从而保证"漏检率"及"误检率"都小于 δ. 但是其假设前提是流抽样要满足"等概率随机抽样". 我们分别使用基于哈希流抽样(简记为 HBFS)和基于 Bloom filter 流抽样(简记为 BFBFS)来实现一级过滤算法. 两种抽样算法都使用 H_3 哈希函数.

我们对不同的流定义分别进行实验, 每次实验根据给定的参数 k_s 在每个背景流中加入 100 个超连接主机发出(或接收)的异常流序列, 每个流序列的流个数为 k_s; 另外再加入流个数小于 k_s/b 的流序列 10000 个以测试"误检率". 以上每个异常流序列的流 ID 首先采用线性流 ID 序列进行实验, 然后再采用随机流 ID 序列重新进行实验. 参数 k_s 取 500、1000、5000 及 10000; 对每种 k_s, b 取 2、5、10; 信任度 $\delta=0.03$.

表 2 是实验结果. 可以看出: 流 ID 序列为线性流 ID 时, 使用新算法(BFBFS)时所产生的"漏检率"及"误检率"都小于其理论值(0.03). 而使用 HBFS 所产生的"漏检率"大部分情况都超过了理论值, 而且比 BFBFS 时的"漏检率"高出 2 至 9 个百分点, 这对于安全应用是很高的. 虽然使用 HBFS 所产生的"误检率"小于理论值, 但是仍比 BFBFS 高出 10 至 40 倍, 这意味着用来维护嫌疑超连接主机的存储空间要大 10 至 40 倍; "误检率"的绝对值虽小, 但是由于正常流数量巨大, 误检所引起的存储空间的消耗也是可观的. 使用 HBFS 时产生较大误差率的原因正如 3.3 节所验证: HBFS 对线性流 ID 序列不能进行等概率随机抽样. 对于随机流 ID 序列的情况, 使用两个抽样算法时的结果都符合理论值. 值得注意的是: 使用 BFBFS 所产生的结果与线性流 ID 序列的情况下的结果没有显著区别, 说明 BFBFS 对线性和随机流 ID 序列都能满足"等概率随机抽样", 具有较好的适应性.

表 2 误检率、漏检率实验结果

k_s	b	Linear Flow IDs				Random Flow IDs			
		False Positives (%)		False Negatives (%)		False Positive (%)		False Negatives (%)	
		HBFS	BFBFS	HBFS	BFBFS	HBFS	BFBFS	HBFS	BFBFS
500	2	1.674	0.076	8	0	0.074	0.077	0	0
	5	1.089	0.027	5	1	0.028	0.027	1	0
	10	2.506	0.248	5	0	0.266	0.251	0	0
1000	2	1.781	0.118	11	2	0.129	0.115	3	1
	5	0.55	0.018	8	1	0.034	0.019	0	0
	10	2.27	0.219	7	0	0.306	0.303	0	0
5000	2	1.977	0.158	6	1	0.191	0.169	0	1
	5	0.75	0.025	6	0	0.024	0.028	1	1
	10	3.913	0.384	4	0	0.358	0.391	0	0
10000	2	1.846	0.183	5	1	0.182	0.192	1	0
	5	0.756	0.021	3	0	0.025	0.024	0	1
	10	3.783	0.356	2	0	0.374	0.345	0	0

5.3 "及时性"评价

与上节的实验类似, 仍然对不同的流定义分别进行实验, 每次实验在背景流中加入若干超连接主机发出(或接收)的流序列, 每个流序列的流个数均为 65535. 流 ID 序列采用线性流 ID 序列和随机流 ID 序列两种形式. 对不同的抽样概率 r (取 10、32、100、500、1000), 我们分别使用基于哈希流抽样算法(HBFS)和基于 Bloom filter 流抽样算法(BFBFS)进行抽样. 当一个流被抽样时, 若它是一个超连接主机所发出(或接收)的流中第一个被抽样的流, 则记录其在该超连接主机所发出(或接收)的流中的位置, 即 n_s. 最后, 分别统计两个算法在不同的抽样率下 n_s 的均值 $E(n_s)$.

图 8 是采用线性流 ID 序列时的结果. 可以看出: 对于不同的抽样率, 基于 Bloom Filter 流抽样算法的结果符合理论值(即 $1/r$). 而基于哈希流抽样算法的结果值均高于理论值, 例如在抽样率为 0.1 ($1/r=10$) 时, 其结果值为 200 多, 20 倍于理论值. 这意味着将会需要通过更多攻击流才能检测出超连接主机. 在安全应用中, 反应速度是衡量检测系统的一个重要指标. 如果反应速度过慢, 即使检测系统能检测出入侵事件的发生, 也会由于

所"放过"的攻击数据太多而造成较大的损失. 流 ID 序列采用随机流 ID 序列时的结果如图 9, 两个算法的结果均符合理论值.

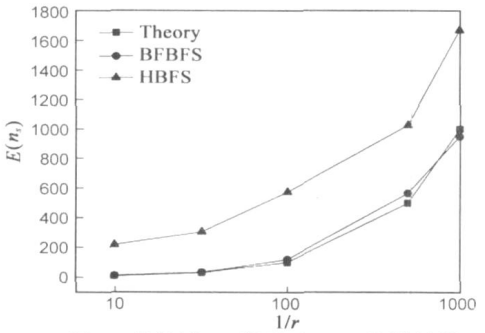

图 8　线性流 ID 序列时 $E(n_s)$统计结果

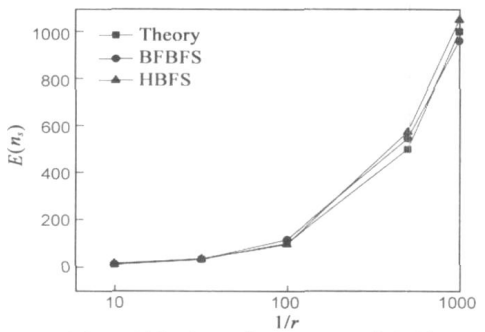

图 9　随机流 ID 序列时 $E(n_s)$统计结果

6　结论

在高速网络环境下, 快速、准确地检测超连接主机是网络入侵检测中的重要问题. 而流抽样是解决其可扩展性的基础. 现有研究都采用基于哈希流抽样算法. 本文首先总结了流抽样的本质需求和哈希函数的选择因素, 并通过技术分析和实验验证指出: 基于哈希流抽样算法不能有效地扩展到更高速的网络环境下(2. 5Gbps 以上) . 针对这一问题, 本文提出了一种基于 Bloom filter 流抽样算法并进行了算法分析和实验验证, 结果表明: 新算法能够通过引入较小的空间代价而扩展到 10Gbps 高速网络环境中.

有必要指出, 新算法的设计并非试图完全取代已有算法. 二者各有优缺点, 各自适用于不同环境. 基于哈希流抽样适用于 1Gbps 左右速率的网络中, 此时可以使用加密性哈希函数以获得理想的等概率流抽样, 算法可以软件或硬件实现, 成本较低. 基于 Bloom filter 流抽样算法适用于 2. 5Gbps 至 10Gbps 的高速网络环境下, 但需要专用的硬件设计和一定量的高速存储器, 成本高于前者. 系统设计及实现者需要根据网络环境、性能要求、成本预算等具体因素在两个算法之间进行取舍.

致谢　美国 University of California(San Diego) 的 Larry Carter 教授关于 Universal 哈希函数对本文第一作者提供了无私的帮助, CAIDA 及 NLANR 为我们的实验提供互联网数据, 对此表示衷心感谢!

参考文献:

[1] Paxson V Bro. A system for detecting network intruders in real-time[J]. In Computer Networks, 1999, 31(23 - 24): 2435 - 2463.

[2] Roesch M. Snort-lightweight intrusion detection for networks [A]. In Proceedings of the 13th USENIX Systems Administration Conference[C]. Washington: USENIX, 1999. 229 - 238.

[3] Plonka D. FlowScan: a network traffic flow reporting and visualization tool[A]. In Proceedings of the 14th USENIX conference on System administration[C]. New Orleans: USENIX, 2000. 305 - 318.

[4] Keys K, Moore D, Koga R, et al. The architecture of CoralReef: an internet traffic monitoring software suite[OL]. http://www. caida. org/publications/papers/2001/CoralArch/coralreef. pdf, 2001.

[5] Levchenko K, Paturi R, Varghese G. On the difficulty of scalably detecting network attacks[A]. In Proceedings of the 11th ACM Conference on Computer and Communications security (CCS' 04)[C]. New York: ACM, 2004. 12 - 20.

[6] Kompella R R, Singh S, Varghese G. On scalable attack detection in the network[A]. In Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement[C]. New York: ACM, 2004. 187 - 200.

[7] Venkataraman S, Song D, Gibbons P, Blum A. New streaming algorithms for fast detection of superspreaders[A]. In Proceedings of the 12th Annual Network and Distributed System Security Symposium[C]. San Diego: The Internet Society, 2005. 120 - 137.

[8] Keys K, Moore D, Estan C. A robust system for accurate real-time summaries of internet traffic[A]. In Proceedings of the 2005 ACM SIGMETRICS international conference[C]. New York: ACM, 2005. 85 - 96.

[9] Zhao Q, Xu J, Kumar A. Detection of super sources and destinations in high-speed networks: algorithms, analysis and evaluation[J]. IEEE Journal on Selected Areas in Communications, 2006, 24(10): 1840 - 1852.

[10] Wang H, Zhang D, Shin K G. Detecting SYN flooding attacks [A]. In Proceeding of IEEE INFOCOM' 02[C]. New York: IEEE, 2002. 1530 - 1539.

[11] Yaar A, Perrig A, Song D. SIFF: a stateless Internet flow filter to mitigate DDoS flooding attacks[A]. In Proceedings of IEEE Symposium on Security and Privacy[C]. Washington: IEEE, 2004. 130 - 143.

[12] Motwani R, Raghavan P. Randomized Algorithms [M]. New York: Cambridge University Press, 1995.

[13] Donald E K. The Art of Computer Programming volume 3: Sorting and Searching (second edition) [M]. US: Addison-Wesley, 1998.

[14] Duffield N G, Grossglauser M. Trajectory sampling for direct traffic observation [J]. IEEE/ACM Transactions on Networking, 2001, 9(3): 280-292.

[15] Crosby S A, Wallach D S. Denial of service via algorithmic complexity attacks[A]. In Proceedings of the 12th USENIX Security Symposium[C]. Washington: USENIX, 2003. 29-44.

[16] Carter J L, Wegman M N. Universal classes of hash functions [J]. Journal of Computer and System Sciences, 1979, 18(2): 143-154.

[17] Järvinen K, Tommiska M, Skyttä J. Hardware implementation analysis of the MD5 hash algorithm[A]. In Proceedings of the 38th Hawaii International Conference on System Sciences[C]. Hawaii: IEEE Computer Society, 2005. 1-10.

[18] Ramakrishna M, Fu E, Bahcekapili E. Efficient hardware hashing functions for high performance computers[J]. IEEE Transactions on Computers, 1997, 46(12): 1378-1381.

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

[20] Broder A, Mitzenmacher M. Network applications of bloom filters: A survey[J]. Internet Mathematics, 2004, 1(4): 485-509.

[21] Ramakrishna M V. Practical performance of bloom filters and parallel free-text searching[J]. Communications of the ACM, 1989, 32(10): 1237-1239.

[22] Xilinx Inc. Virtex-5 Family Overview: Advanced Product Specification[OL]. http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf, 2007.

[23] Huber J. Design of an OC-192 flow monitoring chip. San Diego[R]. University of California, San Diego Class Project, 2001.

[24] Dipert B. Special Purpose SRAMs smooth the ride[OL]. http://www.edn.com/contents/images/45963.pdf, EDN, Jun 24, 1999.

[25] Cooperative association for internet data analysis (CAIDA) [OL]. http://www.caida.org/, 2007.

[26] National Laboratory for Applied Network Research (NLANR) [OL]. http://pma.nlanr.net/, 2007.

作者简介:

王洪波 男, 1975 年 10 月出生于河北, 博士, 北京邮电大学计算机科学与技术学院讲师. 主要研究方向为互联网测量与管理、P2P 技术等. E-mail: hbwang@bupt.edu.cn