计算机科学 2006Vol. 33№. 4

一种动态哈希树网络流量跟踪算法

汪文勇　黄鹂声
（电子科技大学软件工程研究中心　成都 610054）

摘　要　本文提出一种基于动态哈希树的流量跟踪算法 DHT(Dynamic Hash Tree)。该算法利用网络会话的长时稳定性，动态搭建一个由多哈希表组成的树，以提高实际网络环境中会话识别和流量跟踪的速度。试验结果表明该算法的效率明显优于目前流行的哈希链表算法，能够满足骨干网络的实时监测要求。

关键词　流量跟踪，网络会话，长时稳定性，哈希树

A Dynamic Hash Tree Algorithm for Network Flow Tracing

WANG Wen-Yong　HUANG Li-Sheng
(Software Engineering Research Center, UEST of China, Chengdu 610054)

Abstract　A flow tracing algorithm based on dynamic hash tree (DHT) is presented, which takes use of the long-term stability of network session, constructs a dynamic tree with multiple hash tables, and can speed up the network session identifying and flow tracing process. Test results verify that the efficiency of the algorithm evidently surpass the prevailing hash chain table algorithm.

Keywords　Flow tracing, Network session, Long-term stability, Hash tree

1　流量跟踪问题分析

网络测量已经成为了网络管理和运行维护的一个重要基础。在高速网络和复杂应用的背景下，如何实现对网络会话上下文的实时测量与分析是一个需要解决的问题。

流量跟踪是实现上下文分析的基础，它对各个 UDP 或 TCP 连接的协议状态、协议参数进行跟踪提取和上下文关联，并采用专用数据结构记录其结果，作为进一步分析和统计的基础。例如目前主流防火墙的状态检测技术，就需要采用一个会话表来跟踪和维护网络流量的协议上下文(即会话状态)。

本文分析了现有流量跟踪算法的不足之处，并根据流量的本地性，提出"动态 Hash 树"流量跟踪算法(Dynamic hash tree，本文简称为 DHT 算法)，并给出实验结果。

2　流量跟踪算法

2.1　算法定义

假设骨干网上有 N 个活动网络会话，每个网络会话可以用 5 元组(源地址 SA、目的地址 DA、源端口 SP、目的端口 DP、协议 PT)来进行标识。流量跟踪需要为每个网络会话保持一个相对固定的数据结构(Connection Data，简称为 CD)，用于记录其协议状态和实时分析结果。

流量跟踪算法过程，需要快速地将每一个分组定位到它所归属的 CD 数据结构里面，以完成上下文分析。该过程其实就是从 5 元组 Conn(SA, DA, SP, DP, PT)到 CD 集合 S {CD1, CD2, …, CDn}中某个元素的映射和定位过程。一个好的算法，要求占用内存空间少、速度快，而这两个因素往往相互制约。

2.2　现有流量跟踪算法

最简单的流量跟踪算法是采用会话链表。该算法为每个网络会话分配一个链表节点(node)，每个 node 分别保存"上一节点"和"下一节点"两个指针，将所有 node 串联在一起，形成一个双向链表。对每个分组，都需要按照 5 元组顺序检索链表(最坏的情况下需要遍历整个链表)，才能找到相应的 node。该算法空间复杂度为 O(N)，占用内存较小，缺点是时间复杂度较高，需要大量的检索和比较过程。

目前广泛使用的算法是 Hash 链表[1,2]。该算法的思想是：建立一个固定大小的 Hash 表(如 64k 的 Hash 表)，Hash 表中每个表目(Bucket)都指向一个"冲突链表"(impact chain，简称 IC)，以解决哈希冲突问题，冲突链表中每个 node 分别记录了一个网络会话的跟踪数据。

该算法对每个分组都提取 5 元组 Conn(SA, DA, SP, DP, PT)，进行数值累加、折叠等哈希运算，得到一个 0～65535 之间的 Hash 值。然后根据该 Hash 值，顺序检索 Hash 表目对应的冲突链表 IC，以找到该分组所属的网络会话。

由于互联网的特性，实际环境中的 IP 地址和端口往往会集中于某一局部范围，采用简单的 Hash 算法很难保证 Hash 表目被均匀地利用，即使是较好的 Hash 算法，也会造成 O(LOG(N))程度的涨落[3](好的算法由于复杂度高，本身也会造成额外 CPU 开销)；而且，由于部分网络会话超时时间很长，导致局部子网中网络会话数较大，蠕虫病毒和黑客攻击也会产生大量虚假的局部网络会话，所以部分表目产生 Hash 冲突的可能性很大，导致冲突链表 IC 长度增加，检索性能急剧下降。

一种改善的方法是不处理无状态的分组，用以减少会话条目，但该方法存在监测遗漏问题[4]。采用加大表目来提高哈希表查找速度，又会消耗太多的内存。有人曾提出了利用 IP 流的本地性(部分本地网络会话产生的流量是网络中的关键流量，这些会话的 IP 分组在短时间内多次到达的概率较大)来优化查找和分类过程[5～8]，将最活跃的网络会话提升到

链表 IC 的头部，以尽量降低链表检索开销，并将流量较高的会话记录放入高速 Cache。以上方法在活动连接较少的局部网络环境下，可以加快流量跟踪过程，但不能解决根本问题，即无法减少冲突链表 IC 的深度，因而无法适应存在大量高速并发连接的高速骨干网络环境。

3 网络会话的长时稳定性

经过长期对骨干网络的统计和测量，我们发现，互联网中的网络会话分布是不均匀的，而且在不同的时段，网络会话的的产生和分布具有一定的规律，在较长的时间内，这种规律能够得以稳定保持。

将被监测网络内的当前活动会话数记为 S，每秒产生的新会话数记为 ST，其中某个子网的活动会话数和每秒产生的新会话数分别记为 Sn 和 STn。则网络会话的长时稳定性体现在几个方面：

(1) 总会话数：一段时间内，一个网络内的 S 趋于稳定。

(2) 会话数的 IP 子网分布：一段时间内，一个网络内的 S 大多来自于某些子网，这些子网的 Sn 在 S 中占有恒定比例。

(3) 单位时间产生的新会话数：一段时间内，一个网络内的 ST 和 STn 趋于稳定。

(4) 总会话数与单位时间内新会话数的关系：S 和 Sn 呈线性关系，且具有基本稳定的比值。

我们在 CERNET 上对以上规律进行了验证，监测的网络为包含了 30 个 C 类地址的校园网络主干，并随机选择了 2 个子网进行会话分布统计，在 8 小时内，选取了 5 个监测时间窗口，结果如表 1 所示。

表 1 网络会话测量统计结果

时间窗口	S	ST	Sn	STn	S/ST	Sn/S
窗口 1	112151	685	6313	32	163	5.6%
窗口 2	112206	691	6219	33	162	5.5%
窗口 3	114998	714	6401	38	161	5.6%
窗口 4	119688	729	6499	45	164	5.4%
窗口 5	120810	733	6487	40	164	5.3%
平均值	115970	710	6383	38	163	5.5%

根据以上规律，我们可以根据网络在一段时间内的会话统计数据，来预测将来一段时间内网络会话的数量，并对会话的 IP 子网比重分布进行预见，由此可以构造出更加符合实际环境的流量跟踪数据结构。

4 DHT 算法设计

根据 IP 流的本地特性，网络流量可视为由“本地网络流量”和“远程网络流量”组成，研究表明绝大部分网络流量其实是由少部分“本地网络”上的主机产生的[7]。DHT 算法利用以上特性，首先按照双方 IP 子网的不同，建立静态的“根 Hash 表”(Root Hash，简称 RH)以统计各个子网间的流量；然后依据网络会话的长时稳定性，学习实际环境流量，为一些活动频繁、网络会话数较多的本地子网生成动态的子 Hash 表(Child Hash，简称 CH)，形成一个具有明显本地化特征的动态 Hash 树，避免某个 RH 表目对应的冲突链表 IC 过长。

DHT 算法通过增加 Hash 查找过程，减少冲突链表深度，以提高会话的查找速度，改善流量跟踪效率。算法思想如图 1 所示。

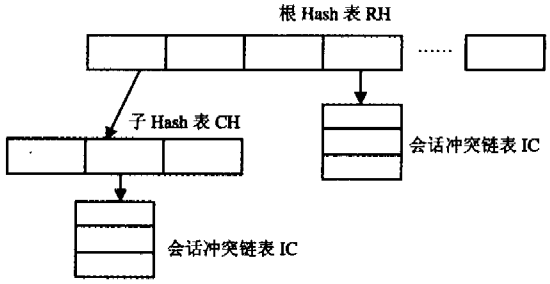

图 1 DHT 算法思想

DHT 算法通过以下 4 个步骤来完成：

(1)建立 RH

算法首先建立静态的 RH，大小为 1k，Hash 值为双方 IP 地址中某些位累加的结果(根据实际情况不同，可以调整位的选择窗口，本文取第 14～23 位用于大致标识两个 IP 子网之间的流量)。根 Hash 函数定义为：

$$F(SA, DA) = ((SA \gg 8) \&0x03FF + (DA \gg 8) \&0x03FF) \&0x03FF \quad (1)$$

所有网络会话都可以通过一次 Hash 运算，被定位到 RH 内的某个表目，并在表目对应的会话链表中保持一个 node。每个 RH 表目中包含三个字段：冲突数量(impact number，简称 IN)、指向下级 Hash 表 CH 的指针(Pointer to CH，简称 PCH)、指向冲突链表的指针(Pointer to IC，简称 PIC)、指针描述符(Pointer Descriptor)。

(2)RH 学习

一旦 RH 被建立后，就可以在实际环境中进行学习，学习目的是为各个表目填写冲突数量字段 IN。由于网络会话的稳定性，我们可以根据在一段时间内在某个表目中新增的网络会话数量(aps)，来估算正常状态下各个 RH 表目所应有的冲突数量。计算公式为：

$$IN = aps \times pat \quad (2)$$

其中 aps 定义为每秒命中某个 RH 表目的新网络会话，pat 为经过统计学习得到的网络会话平均持续时间长度，以秒为单位。经过试验证明，RH 学习时间越长，得到的 aps 值越趋于稳定，而 pat 值在经过一个稳定期后会缓慢增长(由于部分超时或半开会话连接的存在)。得到的 IN 值表示该 RH 表目应该能够维护的平均并发会话数量，当某个 RH 表目的 IN 值较大，则说明该表目是“高冲突点”，将会持续产生大量并发网络会话，应该参考 IN 值为其建立相应的子 Hash 表 CH；反之，则说明该表目利用频度不高，可以直接挂接冲突链表，而不必为其建立 CH。

(3)建立 CH

当 RH 通过一段时间的学习，绝大部分表目的 IN 值趋于稳定的时候，就可以为 IN 值比较大(超过某个阈值 g)的表目建立 CH 了。

考虑到网络上存在大量的长时会话和半关闭会话，一个 RH 表目对应的 CH，其大小应该大于该表目所记载的 IN 值。CH 采用双方端口与 IP 地址低位混合的结果作为 Hash 值，Hash 函数定义为：

$$F(SA,DA,SP,DP) = (SA\&0x00FF \times DA\&0x00FF + SP\&0x00FF \times DP\&0x00FF) \& (length) \quad (3)$$

其中 length 值就是 CH 的大小，由本 CH 对应 RH 表目的 IN 值乘以调整因子而来：

$$length = IN \times adjust \quad (4)$$

根据环境的不同，adjust 有所变化，经过在 CERNET 网上的反复试验，adjust 取值为 6～8 比较合适。

(4)建立与维护 IC

通过以上三个步骤，由 RH 和 CH 构成的 Hash 树已经能够在较大程度上减少 Hash 冲突，但 Hash 冲突仍然存在，解决方式仍然是采用冲突链表 IC。

DHT 算法对 IC 链表的使用过程有几个特点：对新建立的会话，直接添加到 IC 链表的头部，加快其检索过程；在深度>4 的情况下，按照访问频度的高低，逐步将访问频度高的节点提升到链表头部；定时对链表进行清理，清除长时间不活动的会话节点，回收内存空间。

在一段时间以后，网络的流量比重可能会发生变化，此时 RH 的学习过程会识别到这一比重变化(部分 RH 表目的 IN 值在一段时间内持续增长)，并对该表目进行动态扩展，以适应新的流量特征。

总结起来，DHT 算法的核心思想是：通过根 Hash 表的学习和统计，尽量识别出网络中各类关键流的会话比重；通过二级 Hash 表的扩展，尽量使关键流平均分布到有限的 Hash 表目中去，以尽量小的空间争取尽量少的 Hash 冲突，从而减少链表检索。

DHT 算法的时间复杂度为 $O(M+2)$，空间复杂度为 $O(N)$，M 为冲突链表的深度，N 在最小情况下为 1k，最大情况下为 1k+adjust×实际网络会话数。从理论上分析，IC 链表的理想深度应为 1，最大深度为 $\log(K)$，K 为实际环境中最大关键流的并发会话数量。

在一般情况下，深度为 2 的 Hash 树已经能够满足骨干网络环境下数十万条并发网络会话的跟踪要求，因此本文不对 CH 进行进一步扩展。

5 实验与验证

为了验证算法有效性，以 DHT 算法和 Hash 链表算法分别构建测试系统，进行对比运行观测，网络环境为峰值流量 800 Mbps 的校园网出口，硬件平台为 2.4G CPU 的 PC 服务器，内存为 2G，操作系统为 Windows XP Professional，实验结果如图 2 所示。

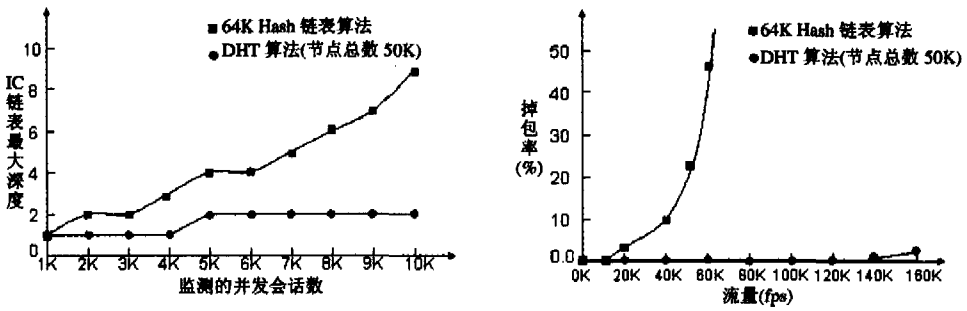

图 2 DHT 算法与 Hash 链表算法对比

图 2 的结果表明，DHT 算法的空间复杂度和时间复杂度均优于 Hash 链表算法，由于 Hash 树的构造符合网络实际环境，因此在网络并发会话数量较高的情况下，可以降低 IC 链表检索开销。在骨干网络环境下，丢包率很低，其总体效率是传统算法的数倍。

结论 DHT 算法充分利用了 IP 流的本地性特征，通过学习和动态构造过程对 Hash 表进行了重新组织，能显著提高 IP 会话的检索性能，测试结果也证实了该算法的有效性。

参 考 文 献

1 Decasper D, Dittia Z, Parulkar G, et al. Router plugins: asoftware architecture for next generation routers [J]. IEEE/ACM Transactions on Networking, 2000, 8(1): 2～15

2 Rusty R, Harald W. Linux Netfilter Hacking HOWTO [EB/OL]. http://www.netfilter.org, 2003-03-09

3 Stoica I, Morris R, Karger D, Kaashoek M F, Balakrishnan H. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. Annual Conference of the Special Interest Group on Data Communication (SIGCOMM 2001). Aug. 2001

4 Gill S. Maximizing firewall availability: techniques on improving resilience to session table DoS sttacks[EB/OL]. http://www.gorbit.net/, 2003-02-23

5 Jain R, Routhier S A. Packet trains: measurements and a new model for computer network traffic [J]. IEEE Journal on Selected Areas in Communications, 1986, 4(6): 986～995

6 Feldmeier D C. Improving gateway performance with a routing table cache [A]. In: Proc. of IEEE INFOCOM [C]. New York: IEEE, 1988. 298～307

7 Xu J, Singhal M, Degroat J. Novel cache architecture to support layer four packet classification at memory access speeds [A]. In: Proc. INFOCOM 2000 [C]. Piscataway, USA: IEEE, 2000. 1445～454

8 郑卫斌，段中兴，等. 基于 IP 流本地性的状态检测性能优化方法. 西安交通大学学报[C], 2004, 38(4)