一种高并发网络环境下快速流表查找方法（2015 网安年会）

王鹏[1,2,3]，周舟[1,2]，刘庆云[1,2]

（1. 中国科学院 信息工程研究所，北京 100093；2. 信息内容安全技术国家工程实验室，北京 100093；

3. 中国科学院大学，北京 100049）

摘　要：为了改进高速网络环境下连接表的查找速度，利用定量分析法研究了 OC-192 骨干链路上传输层流量局部性特征，其不仅具有高并发和高到达速率的特点，而且在适当的缓存窗口下，具有较好的网络局部性特征。基于这些特征和局部性原理，在朴素的哈希表结构基础之上增加常量的辅助空间，实现一种高效流量局部性聚合方法以及快速哈希表查找方法，有效降低了高并发网络中连接表的平均查找长度，提高了流处理系统效率和稳定性。

关键词：哈希表；高并发网络；连接表管理；网络局部性；

中图分类号：TP302　　**文献标识码：**A　　**文章编号：**

A Faster flow table lookup for high concurrency network

Wang Pang[1,2,3], Zhou Zhou[1,2], LIU Qing-yun[1,2]

(1. Institute of Information Engineering, Chinese Academy of Science, Beijing 100093, China;

2. National Engineering Laboratory for Information Security Technologies, Beijing 100093, China;

3. University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: In order to improve connection table lookup speed, flow locality characteristics of the transport layer on the OC-192 backbone links was explored by quantitative analysis method , which not only has high concurrency and high arrival rate of features, but has a good network locality characteristics in an appropriate cached window. Based on these characteristics and the principle of locality , an efficient and rapid locality polymerization process was achieved on naive hash table structure with constant increase of auxiliary space , and a faster hash table lookup method was implemented to reduce the average length of the connection table lookup, finally improved the efficiency and stability of the stream processing system .

Key words: hash table; high concurrent network; flow management; network locality

1　引言

在高速网络环境下，高效率的连接管理已经成为现有网络流量处理系统（如入侵检测、流量计费等系统）的一个关键模块[1][2]，如图 1 所示，通常流量处理系统架构分主要分为三大模块：流量获取、连接管理、业务处理。连接管理为业务处理提供流追溯功能，包括查找、更新和删除这三种操作，为了准确的记录每一条连接，连接管理模块必须维护一个连接表（或会话表），其中每一个连接表项追溯网络中的一条连接，负责记录连接的标识 ID、状态等相关信息，其中连接标识是全局唯一的，一般由 TCP/IP 头部的五元组构成：FID {sip, dip, sport, dport, ip protocal}。

现代连接管理模块所采用的数据结构大多为哈希表，哈希冲突通过连接法解决[3][5][8]。因为在网络中流总数和表长度相近时，哈希表能够支持接近常量时间复杂度的查找、插入、删除操作[4]。随着网络的快速发展，网络带宽不断增加，测量结果表明，网络字节数中 TCP 流量在总量中占比约 78%[10]；骨干网络中并发连接数达到上百万[5]。这对连接管理带来新的挑战：并发连接数的增加导

基金项目：国家自然科学基金资助项目(61402464)；中国科学院战略性先导科技专项（No. XDA06030200）

Foundation Items: National Natural Science Foundation of China (61402464);

The Strategic Priority Research Program of the Chinese Academy of Science(No. XDA06030200);

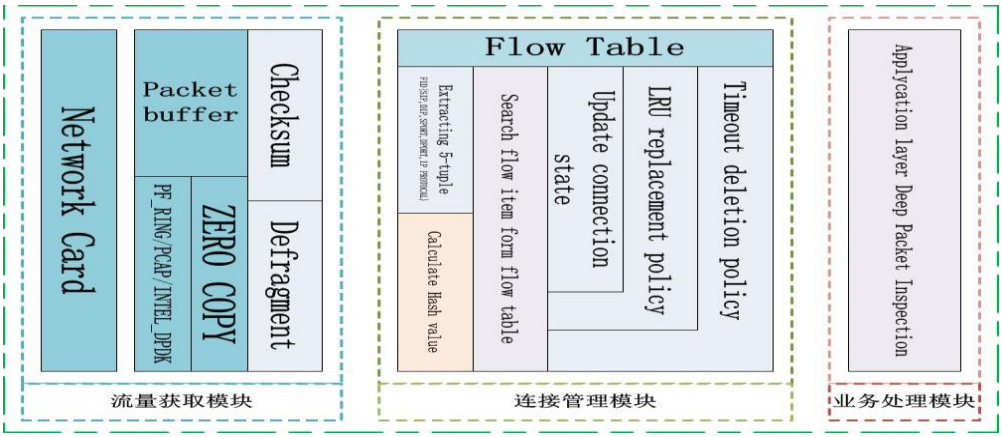

图 1 流处理系统架构示意

致连接表的规模庞大，受限于硬件资源的限制和哈希表结构自身的局限，哈希表槽数需要预先设定且动态调整非常困难[9]，冲突链的增长使得单包流表查找效率下降；在现有 10Gbps 流量的高速网络中，包速达到 10Mpps 甚至更高，而网络中绝大多数包都需要执行流表查找操作，流表的查找频率和包到达速率相当，流表查找效率已经成为流处理系统的重要性能之一。

为了能够加快高并发网络环境下的流表查找操作，我们借鉴局部性原理，首先定量分析了高并发环境下的流量局部性特征；之后在传统哈希表的结构基础之上，增加了常量开销的辅助空间，设计了一种简单高效的网络局部型聚合方法和快速哈希表查找方法，使得平均查找长度比传统方法下降 30%以上，有效减少了高并发环境下的流表查找长度，提高了流处理系统的效率和稳定性。

文章组织结构：第 2 部分描述相关工作；第 3 部分先简单介绍传统连接管理模型，接着对当前骨干网络流量局部性特征进行详细分析；第 4 部分给出局部性聚合方法和快速查找方法的实现步骤以及理论复杂度分析；第 5 部分对新方法进行评测；第 6 部分总结全文。

2 相关工作

目前的流查找操作被分为三类实现方法：哈希表，布鲁姆过滤器，内容寻址存储器[1]。对采用哈希表结构的流表而言，一次哈希表查找包括哈希值的计算和冲突链比较两步操作。用连接法处理哈希冲突的最坏情况性能很差：所有 N 个关键字都被插入到同一个槽中，从而产生一个长度为 N 的链表，这时的最坏查找长度为 O(N)[4]。

基于此很多工作都集中在尽量使得每个槽上的冲突链长度尽量均衡，以保证平均查找长度尽量接近最好情况的 O(1+α)，α为装载因子。要实现这一点需要好的哈希算法，[3]指出尽管可以借助复杂的密码学哈希方法(MD5、SHA-1)来实现哈希表中的所有冲突链长度分布均衡，但是好的哈希函数通常会消耗大量的 CPU，即使哈希运算单元的硬件实现也需要 64 个时钟周期[16]。

相对于前面提到的一重哈希，多重哈希的效果会更好[6]，布鲁姆过滤器就是一种多重哈希，[3]实现了一种变化的布鲁姆过滤器 FCF(fingerprint compressed filter，指纹压缩过滤器)，FCF 的每一个哈希槽包含 d 个大小固定的子表，通过 d 个互相独立的哈希函数，一条连接将会计算 d 次哈希值，最终插入到 d 个子表中最短的一个，但这使得每次查找都要查找 d 个冲突链，在包数密集的骨干网络里将带来较大的查找开销，而且它使用了单个 bit 位进行超时处理，这是一个槽级别的处理方法。虽然 FCF 将 FID 压缩之后可以获得 SRAM 的快速访问优点，但是其扩展性也受限于 SRAM 的容量。

在借助网络局部性优化查找操作方面，文献[2]通过测量指出骨干网络具有高并发的特点，并发量达到两百万，每秒新出现的连接只有 2 万左右，占总并发量的 1%，骨干网络更新缓慢，存在一定程度的局部性，基于此文献[2]使用 FPGA 和 SRAM 实现了流表的高速 Cache 以加快访问速度，受限于 FPGA 的电路复杂性以及 SRAM 的容量限制，其主要针对数据包负载的前几个字节进行处理，专门应用与应用识别系统，并不能解决如 DPI 系统的全数据包负载处理问题。另外 TCAM[15][16] (ternary content addressable memory，一种内容寻址存储器)也被用来加速查找操作。但 TCAM 价格太贵，耗电严重，而且流表的规模同样受到了存储容量的限制，

在高并发网络环境，受流量波动性和突发流量的影响，大量活动连接将会被迫替换掉，导致系统漏检。

3 连接管理与流量特征

本节先简单介绍经典的连接管理方法，给出基本的查找复杂度和表示法；然后对 OC192 链路 2014 年 21 分钟真实流量局部性特征进行分析。

3.1 朴素连接管理方法

如图 2 所示，朴素连接表(NFT，naïve flow table)使用哈希表进行组织：采用数量固定地址连续的数组实现哈希表的哈希槽(Bucket)，并且每一个槽后面都使用一个链表（CL，Collision List）处理哈希冲突。每一条连接用全局唯一的流标号 FID 标识，

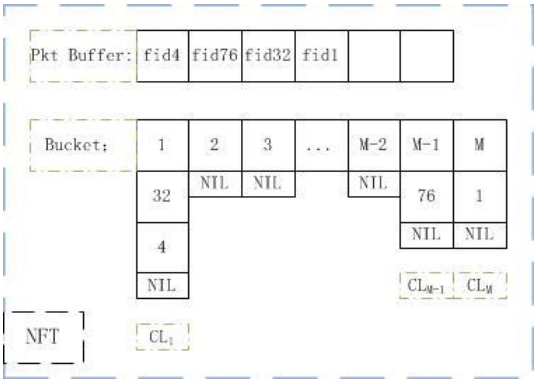

图 2 朴素连接表结构示意

第 i 号流的数据包记为 FIDi。

每当一个数据包 x 到达时，都会按照如下操作进行处理：

过程 1 朴素连接查找

```
1:  fid←getFID(x.sip ,x.dip, x.sport, x.dport. x.ipproto)
2:  hash_value←hash(fid)
3:  j←hash_value%mod
4:  if NFT[j] = NULL then
5:      p←new flow_items(fid)
6:      insert(NFT[j], p)
7:  else
8:      p←search_items（NFT[j],fid）
9:  end if
10: if p=NULL then
11:     p←new flow_items(fid)
12:     Insert(NTF[j], p)
13: end if
14: update(p, x.state)
15: return TRUE
```

其中 search 方法将遍历整个冲突链，逐个比较 FID 信息，决定了整个过程的查找开销，为了说明 NFT 的平均查找长度(NASL，naïve average search length)，定义 NFT 的槽位为数 M, NFT 中已经插入的元素数为 N，则此时 NFT 的装载因子为

$$\alpha = \frac{N}{M} \qquad (1)$$

即平均每条冲突链上的元素个数。由[4]中证明可以知道，在哈希均匀的情况下，一次查找的比较次数期望值为

$$NASL = 1 + \frac{\alpha}{2} - \frac{\alpha}{2N} \qquad (2)$$

由此我们可以确定，在哈希均匀的前提下，流表的平均查找次数和哈希表的槽数成正比，和流表总元素数成反比。

3.2 骨干网络流量局部性特征

局部性指某条连接的一个数据包到达之后，则该连接的数据包将会在不久的将来再次到达。在朴素连接管理中，几乎每个包都要执行一次冲突链遍历，借助局部性，我们可以将同连接的多个数据包集合起来，只查找一次连接表，避免多余的冲突链遍历操作。为此，我们研究了 CAIDA[17]提供的 2014 年 OC-192 链路（14%链路利用率）上 21 分钟的真实流量的局部性特征，数据集已经过匿名化处理。

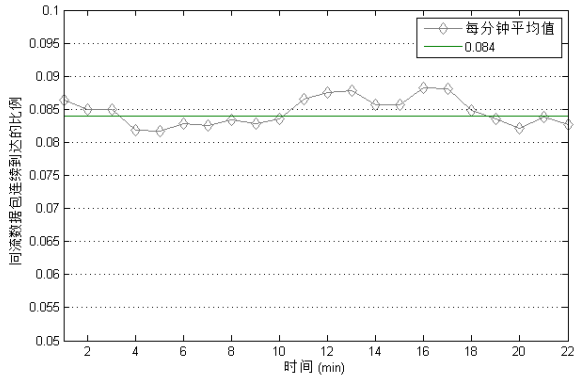

图 3 OC-192 链路局部性

图 3 描述了两个相邻到达的数据包是同一连接的情况占所有数据包的百分比，总共约 21 分钟的流量，每分钟内取平均值。可以看出，由于并发量太大，不同连接的数据包被彼此隔离开，可以观测到的局部性平均为 8.4%。

对于此，我们设计了一种局部性聚合的方法（LP，Locality Polymerization），通过一个缓存窗口，将同一连接的数据包快速分离开，以实现减少流表遍历次数的目标。在这里，我们先给出局部性程度的量化指标，作如下定义：将数据包按照到达顺序排成一个序列 S，对任一长度为 K 的子序列 s：

$s:\left\{x_{a1},x_{a2},\cdots,x_{ai},x_{aj},\cdots,x_{ak}\right\}$，其中 ai 代表流表号，$1\le i\le K$ 。

我们对 s 序列按照流标号进行划分，记总的划分数为 C，表示 s 中共有 C 个不同的连接，令

$$\rho=\frac{C}{K}\ ，\ 显然\rho\in\left[\frac{1}{\mathrm{K}},1\right] \qquad （3）$$

ρ 代表 K 个数据包所包含的连接比例，令

$$\mu=1-\rho \qquad （4）$$

μ 就代表了长度为 K 的包序列的局部性程度。为了更直观的展示网络中的局部性程度，我们在[32，352]的范围内分别取 K 进行了测量。

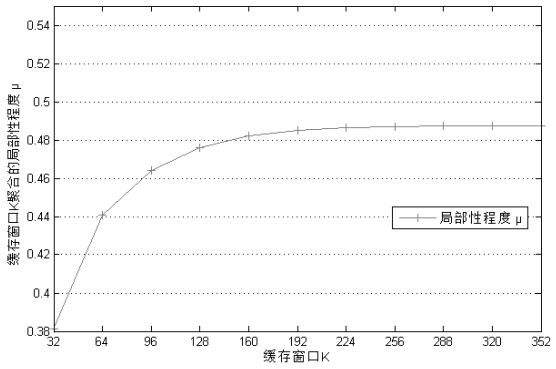

图 4　不同 K 时局部性占总包数的比例

从图 4 可以看出，在 K 取 32 时 OC192 骨干链路的局部性程度达到 35%。由骨干网络并发连接慢更新的特点，结合适当的缓存窗口，确实可以获得较好的局部性。基于这些特征，我们在朴素连接表之上设计了局部性聚合方法（LP），并在此基础上实现了快速连接表查找法。

4　局部性聚合与快速查找

这一章节主要分为两部分，第一部分描述了局部性聚合法、快速查找法的具体实现；第二部分对快速查找法的理论时间复杂度进行分析。

4.1　局部性聚合与快速查找

为了实现局部性聚合，以便利用流量局部性特征，需要一些辅助数据结构，在朴素的哈希表的基础上，增加一个缓冲窗口以及辅助流划分操作的划分表(partition table，PT)。

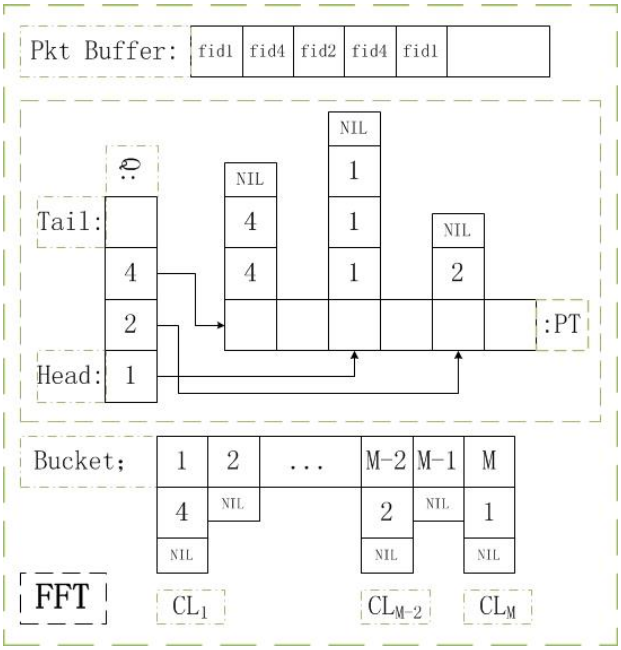

图 5　实现局部性聚合的哈希表结构

如图 5 所示，PT 将不同流的数据包进行了划分，并使用一个固定容量为 K 的队列 Q 对不同的流进行了索引。每当一个数据包 x 到达，都执行如下过程 2 操作：

过程 2　局部性聚合与快速查找

```
1:  fid←getFID(x.sip ,x.dip, x.sport, x.dport. x.ipproto)
2:  hash_value←hash(fid)
3:  j←hash_value%mod_pt
4:  cached_num←cached_num+1
5:  if PT[j] = NULL then
6:      insert x into PT[j]
7:      Q.enque(j)
8:  else if PT[j].first.fid = x.fid then
9:      insert x into PT[j]
10: else
11:     call commit
12:     insert x into PT[j]
13:     Q.enque(j)
14: end if
15: if cached_num=K then
16:     call commit
17: end if
18: function commit
19:     for all j in Q, do
20:         x←PT[j].first
21:         p ←search_items（NFT[j],x.fid）
22:         if p=NULL then
23:             p←new flow_items(fid)
24:             Insert(NTF[j], p)
25:         end if
26:         for all x in PT[j], do
27:             update(p, x.state)
```

```
28:            end for
29:            PT[j] ←NULL
30:        end for
31:        return TRUE
32:    end function
```

由第 11 行和 16 行可以知道，只有在缓存数量达到 K 值或者 PT 表发生冲突时，才会执行 commit 方法，由于 K 值选择相对较小，而 PT 表一般可以选择真实表的 1%到 10%的范围内，冲突的概率极低，不会影响对局部性的利用。

由第 19-21 行可以知道，对于每个 PT[j]划分而言，每次只有 PT[j]链第一个数据包执行真实流表 NFT 的冲突链遍历操作，之后 PT[j]划分中的其他数据包只需要依次进行流状态更新操作即可，此时，一次查找的时间开销将被多个数据包均摊，借此达到降低平均查找长度的效果。而哈希值计算和 FID 抽取在第 1-2 行中已经计算过，不需要重复计算。

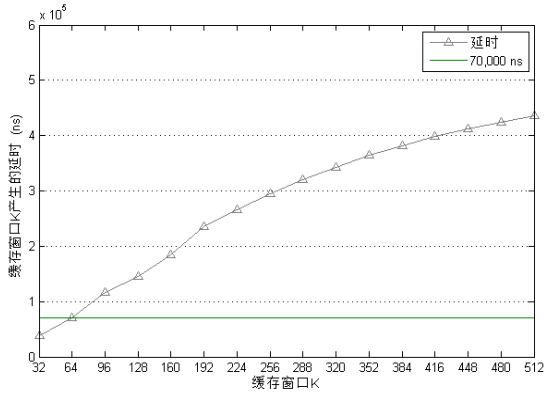

图 6 不同 K 时的延时表现

K值的选择：由于在高速流出里系统中，数据包源源不断的到达，大量缓包是不允许的，缓存窗口选择要折中考虑。因此，在图6中我们对不同的K值带来的平均延时进行了测量，可以看出窗口K取64时的平均延时为70us，局部性程度为44%，已经是一个不错的选择；在K超过96时将导致大于100us的延时，而且K取值更大值所带来的局部性程度也已经趋于上界值，因此在实际选取64是比较恰当的；而且，考虑到流量的突发特性，因为这很有可能导致系统瞬时过载，威胁系统的稳定性。另外，PT的槽数量要尽量大一些，这样保证PT表的低冲突率，而不必过早执行commit方法，可以选择K值的10-20倍之间，比如K取64是PT表槽数可以设为1024。

4.2 快速查找法的复杂度分析

根据 4.1 的操作步骤，我们将快速查找法的平均查找长度（FASL）计算分为两部分之和，划分表的平均查找长度 PASL 和局部性聚合之后的朴素表平均查找长度 $NASL'$。

$$FASL = PASL + NASL' \qquad (5)$$

首先分析 PT 中的查找长度，由于缓存窗口大小为 K，其中存在互不相同的连接数为 $C = \rho K$，在每个连接的第一个包确定后，该连接的其他数据包均需要一次 FID 的比较（对应 4.1 中步骤 2 和步骤 3），即 K 个包中发生 FID 比较操作的次数为：

$$PASL = \frac{K - C}{K} \qquad (6)$$

下面来看 NFT 中的平均查找长度，由于聚合之后，每个连接只需要链首的数据包执行查找操作（对应 4.1 步骤 3），故：

$$NASL' = \frac{C * NASL}{K} \qquad (7)$$

这样， K 个数据包使用快速查找法的平均查找长度为：

$$FASL = \frac{K - C}{K} + \frac{C * NASL}{K} \qquad (8)$$

最终，我们得到平均查找长度和局部性程度之间的关系式：

$$FASL = NASL - \mu * (NASL - 1) \qquad (9)$$

有上式可以看出，FASL 在 NASL 大于 1 的时候始终比 NASL 小，即快速查找法确实可以降低平均查找长度。相对于朴素哈希表而言，快速查找法引入的常量辅助存储空间为 O（K+LT）。

5 性能评测

我们在 CAIDA 提供的 OC-192 链路上 20 分钟真实流量 A 和中国科技网网关处捕获的 20 分钟真实流量 B 两组数据集上进行对比，两组数据集连接并发情况概括如下：

表 1 数据集概要特征

数据集	稳定并发数	每秒新增数	pps
A	170W	1.8W	70W
B	40W	0.6W	21W

我们将超时时间设为 120s，PT 的哈希槽数设置为 1447，缓冲窗口设定为 64，真实流表的哈希槽数设定为 10W。实验过程中统计每秒种的平均查找长度，最终以 100 秒为单位求取平均值。

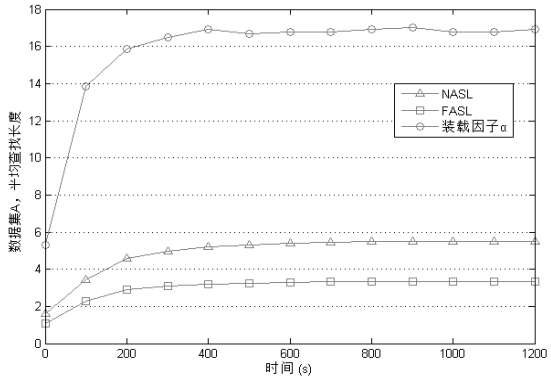

图 7 数据集 A 上平均查找长度对比

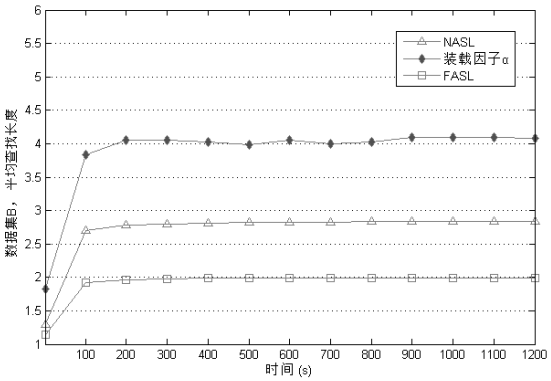

图 8 数据集 B 上平均查找长度对比

从图 7 和 8 中可以看到，快速查找法的平均查找长度在两组数据集上均比比朴素查找法短，查找长度想对于朴素查找法降低了 35%左右。

6 结论

为了改进高并发网络环境下的流表查找开销，本文通过对骨干网流量进行测量，分析高并发网络环境下的流量局部性特征，基于这些特征和局部性原理，提出一种流量局部性量化指标和高效的流量局部性聚合的方法，在此基础上实现了快速流表查找方法，给出了理论复杂度分析。并对 K 值和 PT 槽数选择提出了建议。通过真实流量实验证明，快速查找法加强了流量局部性的适用性，有效的降低了平均查找长度，提高了流处理系统的效率和稳定性。

参考文献

[1] Nam G, Patankar P, Kesidis G, et al. Mass purging of stale tcp flows in per-flow monitoring systems[C]//Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on. IEEE, 2009: 1-6.

[2] Pan T, Guo X, Zhang C, et al. Tracking millions of flows in high speed networks for application identification[C]//INFOCOM, 2012 Proceedings IEEE. IEEE, 2012: 1647-1655.

[3] Dharmapurikar, Sarang, et al. "Fast packet classification using bloom filters." Architecture for Networking and Communications systems, 2006. ANCS 2006. ACM/IEEE Symposium on. IEEE, 2006.

[4] Cormen T H, Leiserson C E, Rivest R L, et al. Introduction to algorithms[M]. MIT press, 2009.

[5] http://libnids.sourceforge.net/

[6] Ayuso P. Netfilter's connection tracking system[J]. LOGIN: The USENIX magazine, 2006, 31(3).

[7] Levandoski J, Sommer E, Strait M. Application Layer Packet Classifier for Linux. 2008[J].

[8] Snort - an open source network intrusion prevention system http://www.snort.org.

[9] Nam G, Patankar P, Lim S H, et al. Clock-like Flow Replacement Schemes for Resilient Flow Monitoring[C]//Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on. IEEE, 2009: 129-136. [10] beyong bloom filters from approximate membership checks to approximate state machines 2006

[10] Zhang, G. "Internet Traffic Measurement and Characteristic Analysis on Output Link of Metro Area Network." Acta Electronica Sinica 35.11 (2007): 2092.

[11] Yoon S, Kim B, Oh J, et al. High performance session state management scheme for stateful packet inspection[M]//Managing Next Generation Networks and Services. Springer Berlin Heidelberg, 2007: 591-594.

[12] House H D L D. HCR MD5: MD5 crypto core family[J]. 2002.

[13] Noureldien N A, Osman I M. A stateful inspection module architecture[C]//TENCON 2000. Proceedings. IEEE, 2000, 2: 259-265.

[14] Fan L, Cao P, Almeida J, et al. Summary cache: a scalable wide-area web cache sharing protocol[J]. IEEE/ACM Transactions on Networking (TON), 2000, 8(3): 281-293.

[15] Meiners C R, Liu A X, Torng E. TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs[C]//Network Protocols, 2007. ICNP 2007. IEEE International Conference on. IEEE, 2007: 266-275.

[16] Lakshminarayanan K, Rangarajan A, Venkatachary S. Algorithms for advanced packet classification with ternary CAMs[C]//ACM SIGCOMM Computer Communication Review. ACM, 2005, 35(4): 193-204.

[17] HTTP://WWW.CAIDA.ORG/DATA/PASSIV/PASSIVE_2014_DATASET.XML