一种高并发网络环境下快速流表查找方法

王 鹏[1,2,3],张 良[4],周 舟[1,2],刘庆云[1,2],方滨兴[1,2,5]
(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;
4. 国家计算机网络应急技术处理协调中心,北京 100029;5. 东莞电子科技大学电子信息工程研究院,广东东莞 523808)

摘 要: 为了改进高速网络环境下连接表的查找速度,本文首先分析了OC-192骨干链路上的流量特征.研究表明,骨干链路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,本文在朴素的哈希表结构基础之上增加常量开销的辅助空间,实现了一种快速流表查找方法.理论分析和真实网络数据集上的实验表明,该方法相比现有方法可以降低流表查找长度20.2%,减少流表访问时间17.1%.

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: TP393 **文献标识码:** A **文章编号:** 0372-2112 (2017)04-0974-08
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2017.04.029

Fast Flow Table Lookup for High Concurrency Network

WANG Peng[1,2,3],ZHANG Liang[4],ZHOU Zhou[1,2],LIU Qing-yun[1,2],FANG Bin-xing[1,2,5]
(1. *Institute of Information Engineering,Chinese Academy of Sciences,Beijing* 100093,*China*;
2. *National Engineering Laboratory for Information Security Technologies,Beijing* 100093,*China*;
3. *University of Chinese Academy of Sciences,Beijing* 100049,*China*;
4. *National Computer Network Emergency Response and Coordination Center,Beijing* 100029,*China*;
5. *Institute of Electronic and Information Engineering,Dongguan UESTC,Dongguan,Guangdong* 523808,*China*)

Abstract: In order to improve flow table lookup speed,flow characteristics of the OC-192 backbone links was explored. It proves that the backbone traffic not only has high concurrency and high arrival rate,but also has good network locality characteristics in an appropriate cached window. Based on these characteristics and the principle of locality,a fast flow table lookup method was implemented by using a naive Hash table structure with constant increase of auxiliary space. The theoretical analysis and experiments on real-life data traces show that the proposed method can reduce the length and the time of flow table lookup by 20.2% and 17.1% compared with the existing method,respectively.

Key words: hash table;high concurrency 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[3],包括查找、更新和删除这三种操作.为了准确地记录每一条连接,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 protocol}.

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[4~6].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[7].现有流量处理系统采用单包调度的策略.数据包首先缓存在网卡缓冲区中,之后按缓冲区内到达顺序依次送往连接管理模块,执行连

收稿日期:2015-03-27;修回日期:2016-05-05;责任编辑:孙瑶
基金项目:中国科学院战略性先导科技专项(No. XDA06030200);国家自然科学基金(No. 61402474);国家242信息安全计划(No. 2015A087);广东省产学研合作项目"广东省健康云安全院士工作站"(No. 2016B090921001)

图1　网络流量处理系统架构

接状态更新与维护操作. 在高速网络环境中,单包处理不仅会带来大量的函数回调开销,也会导致流表访问的性能瓶颈. 随着并发连接数的增加,连接表的规模不断增加. 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[8],冲突链的增长使得单包流表查找效率急剧下降. 在现有 10 Gbps 流量的高速网络中,包速达到 10 Mpps 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量处理系统的重要性能之一.

权威机构 Sandvine 2014 年流量报告[9]指出,亚太地区的骨干链路流量中文件、音视频等多媒体流量占比超过 70%. 同时,骨干链路中,TCP 流量在总量中占比超过 90%[10],流量存在高并发、慢更新的特点,网络流量存在一定程度的局部性[2]. 这意味着在一段时间内,同一连接的多个数据包会陆续到达.

基于骨干链路的流量特征,针对连接管理模块在单包处理时面临的问题,本文首先提出流量局部性量化指标,定量分析骨干链路中流量局部性特征. 其次在传统哈希结构基础之上,增加常量开销的辅助空间,设计实现了一种快速流表查找方法(FFTL),降低流表访问开销. FFTL 分为三个阶段执行:首先通过一个高效的流量局部性聚合算法,对缓冲区内的数据包按流标识分组,使得同一连接的数据包分为一组;其次,利用一种阈值调度策略,对于已经分组的数据包进行调度;最后,根据调度顺序,将每组数据包批量送往连接管理模块,执行流表查找、连接状态维护等操作. 实验结果表明,该方法有效降低了高并发网络中流表的平均查找长度,提高了网络流量处理系统的效率.

2　相关工作

目前的流表查找操作分为三类实现方法:哈希表,布鲁姆过滤器[11],内容寻址存储器[12]. 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,即所有 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 $O(N)$[7].

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 $O(1+\alpha)$,α 为装载因子. 要实现这一点需要好的哈希算法,文献[4]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[13].

相对于前面提到的一重哈希,多重哈希的效果会更好[14],布鲁姆过滤器(Bloom filter)就是一种多重哈希,通常结合高速硬件(TCAM、FPGA[15]等)提供高效内容查找[16]. 文献[17,18]指出 Bloom filter 常用与表示一个集合以提供成员查询操作,其查询结果存在误差概率. 而且 Bloom filter 查询结果只有是与否两个值,不满足流量处理系统中连接状态维护的需要. 文献[4]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前几个字节进行处理,专门应用于协议识别系统,并不能解决如 DPI 系统全数据包负载处理问题. 另外 TCAM(Ternary Content Addressable Memory,内容寻址存储器)[12,19]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存储容量的限制,在高并发网络环境中,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　传统连接管理方法与流量特征分析

本节首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;然后对真实流量局部性特征进行分析.

3.1　朴素连接管理方法

朴素连接表(NFT,Naïve Flow Table)使用哈希表进行组织,如图 2 所示,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,第 i 号流的数据包记为 FIDi.

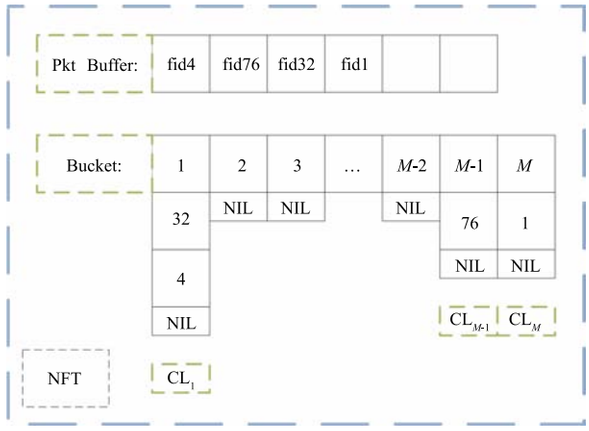

图2 传统连接管理结构

每当一个数据包 x 到达时,都会按照算法 1 进行处理.

算法 1 朴素连接表查找

```
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

$$\alpha = \frac{N}{M} \tag{1}$$

即平均每条冲突链上的元素个数. 由文献[7]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

$$NASL = 1 + \frac{\alpha}{2} - \frac{\alpha}{2N} \tag{2}$$

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,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的遍历. 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[20] 提供的 2014 年 OC-192 链路(14%链路利用率)上真实流量的局部性特征,该数据集已经过匿名化处理.

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%.

基于流量特征,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组,以实现减少流表遍历次数的目标. 我们首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:

$$s:\{x_{a1},x_{a2},\cdots,x_{ai},x_{aj},\cdots,x_{ak}\},$$

其中 ai 代表流标号,$1 \leqslant i,j \leqslant K$.

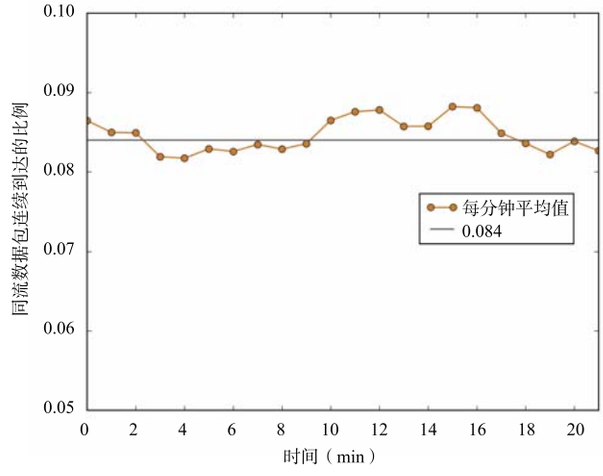

图3 OC-192链路局部性

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

$$\rho = \frac{C}{K}, \text{显然 } \rho \in \left[\frac{C}{K},1\right] \tag{3}$$

ρ 代表 K 个数据包所包含的连接比例,令

$$\mu = 1 - \rho \tag{4}$$

μ 代表长度为 K 的包序列的局部性程度. 为了更直观地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,如图 4 所示.

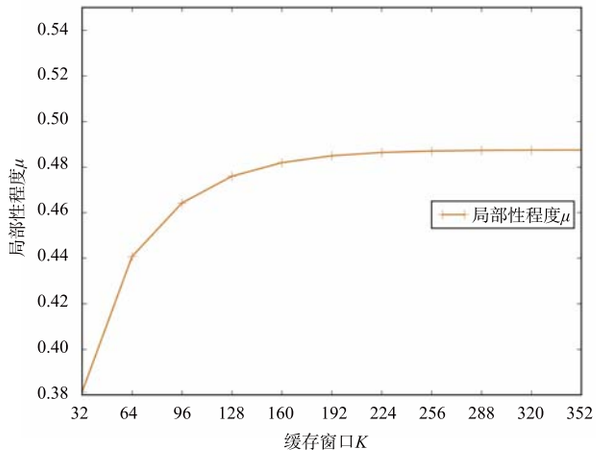

图4　不同K时局部性占总包数的比例

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%. 由骨干网高并发、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

本节首先描述了局部性聚合法、快速查找法的具体实现,同时对聚合后的调度策略进行了讨论和改进;然后对快速查找法的理论时间复杂度进行了分析.

4.1　局部性聚合与调度策略

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),并使用一个队列 Q 对已经完成划分操作的分组进行索引.

如图 5 所示,PT 将不同流的数据包进行划分,并且索引队列 Q 的大小设定为与缓冲区大小相等的量 K. 每当一个数据包 x 到达时,都执行如算法 2 操作.

算法 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:         ptsearch_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:     Q.clear()
32:     return TRUE
33: end function
```

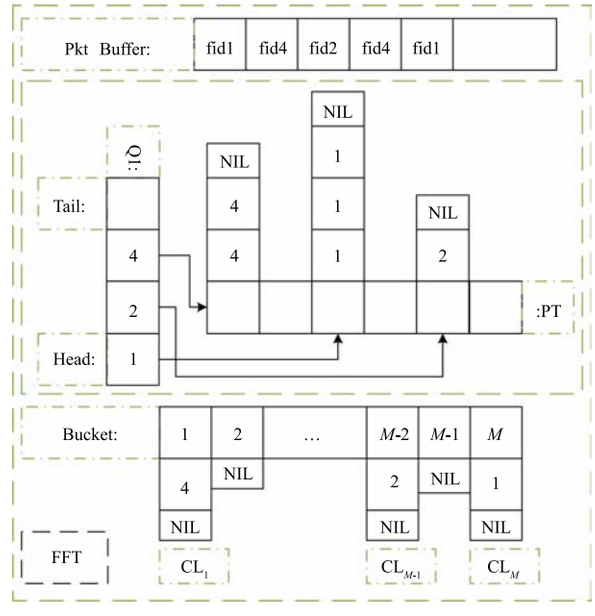

图5　实现局部性聚合的哈希表结构

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.

调度策略的进一步改进:由算法 2 第 19 行可以知道,每次发生提交时,会将索引队列中所有已缓存的连接送往连接管理模块. 由于连接到达的不均匀性,存在一部分连接只缓存一个数据包也被提交的情况,由 3.2 节中(3)(4)可知,减少这种情况的发生可以提高聚合的效果. 由此,我们对 18 – 32 行的 commit 方法引入阈值清空策略,每次提交优先考虑缓存包数达到历史平均值的连接. 对图 5 中局部性聚合模块做如下调整:①增加二级索引队列 Q2,当一级队列 Q1 中某个连接缓存包数达到阈值,则将该连接转移到二级索引队列中进行索引;②将 Q 与 PT 之间的单向索引关系修改为双向索引关系;③每次优先提交 Q2 中的链接,会导致优先级较低的 Q1 中的连接长时间得不到调度机会,为此引入饥饿避免机制,增加冲突计数器 C1 和 Q2 调度累加器 C2. 调整后如图 6 所示.

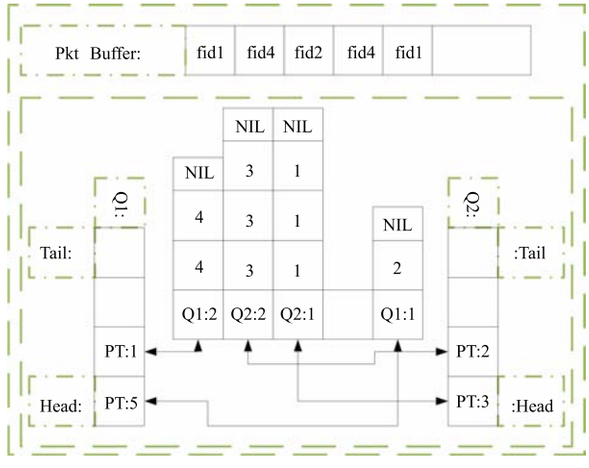

图6 引入二级索引队列的局部聚合结构

当聚合操作发生冲突时,C1 增加 1,当缓冲窗口满且 C1 大于 0 时 C1 减少 1,C2 用于统计 Q2 累积提交的包数;当缓存区满或者发生冲突时首先提交 Q2 中的连接,当 Q1 中包数达到 K 或者 C1 达到阈值或者 C2 达到阈值这三个条件有一个满足时,提交 Q1 中的连接,同时清空计数器. 通过这种方法改进,既可以提高聚合效果,又可有效避免 Q1 的调度饥饿现象. 具体如算法 3 所示.

算法 3 改进后的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:     Q1.enque(j)
8:  else if PT[j].first.fid=x.fid then
9:     insert x into PT[j]
10:       if PT[j].size > delt then
11:            move PT[j] from Q1 to Q2
12:       end if
13:  else
14:       move PT[j] from Q1 to Q2
15:       call commit(Q2)
16:       C1← C1 +1
17:       insert x into PT[j]
18:    Q1.enque(j)
19:  end if
20:  if cached_num=K then
21:     C1← max(C1-1,0)
22:     call commit (Q2)
23:  end if
24:  if Q1.pkt_size=K or C1 > delt or C2 > 10*Q1.pkt_size then
25:       call commit (Q1)
26:     C1←1
27:     C2←20
28:  end if
29:  function commit
30:       if Q is Q2 then
31:            C2←C2 +Q2.pkt_size
32:       end if
33:       for all j in Q,do
34:          x←PT[j].first
35:          ptsearch_items(NFT[j],x.fid)
36:          if p=NULL then
37:               p←new flow_items(fid)
38:               Insert(NTF[j],p)
39:          end if
40:          for all x in PT[j],do
41:               update(p,x.state)
42:          end for
43:          PT[j]←NULL
44:       end for
45:       Q.clear()
46:       return TRUE
47:  end function
```

4.2　快速查找法的复杂度分析

根据4.1小节的操作步骤,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,即划分表的平均查找长度PASL和局部性聚合之后的朴素链接表平均查找长度NASL′.

$$FASL = PASL + NASL' \tag{5}$$

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小节中步骤2和步骤3),即K个包中发生FID比较操作的次数为:

$$PASL = \frac{K-C}{K} \tag{6}$$

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小节步骤3),故:

$$NASL'' = \frac{C * NASL}{K} \tag{7}$$

这样,K个数据包使用快速查找法的平均查找长度为:

$$FASL = \frac{K-C}{K} + \frac{C * NASL}{K} \tag{8}$$

最终,我们得到平均查找长度和局部性程度之间的关系式:

$$FASL = NASL - \mu * (NASL - 1) \tag{9}$$

由式(9)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　实验结果与分析

为了验证FFTL的有效性,我们采用流表平均查找长度和平均访问时间两个评价指标,将FFTL的实验结果和开源入侵检测系统snort的连接管理模块进行对比评测. 按照数据集的流量来源和持续时间我们将实验场景做如下分类.

表1　数据集概要特征

场景	来源	平均bps	最大bps	持续时间
C1	OC-192	6.55G	7.2G	20min
C2	学校网关	9.2G	9.4G	20min
C3	学校网关	9.1G	9.6G	4h

实验采用的系统环境为Redhat 6.3,kernel 2.6.32-279,CPU为Intel Xeon 8核1.8GHz,内存32GB,网卡为Intel 82599万兆网卡.

场景一

由于C1数据集中只包含了IP和TCP头部信息,没有其他载荷数据,无法进行时间维度的评测,因此仅进行了查找长度的实验.

图7对比了Snort连接管理模块、FFTL、引入阈值调度策略的FFTL(MFFTL)在C1中的平均查找长度,其中FFTL和MFFTL的K取值为256. 结果表明快速查找法可以有效的降低流表平均查找长度. 图8则对比了MFFTL在K分别取64、128、256时的平均查找长度,随着窗口增大,会有一定程度的下降,考虑到K值的增加会导致数据包的处理延时增加,缓存窗口选择需要结合系统延时容忍度折中考虑.

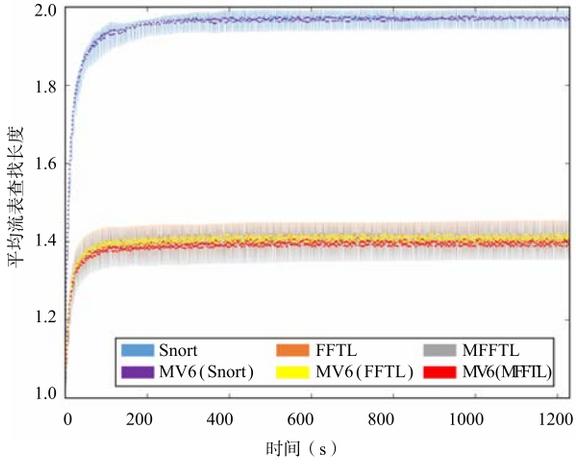

图7　C1场景下流表查找长度

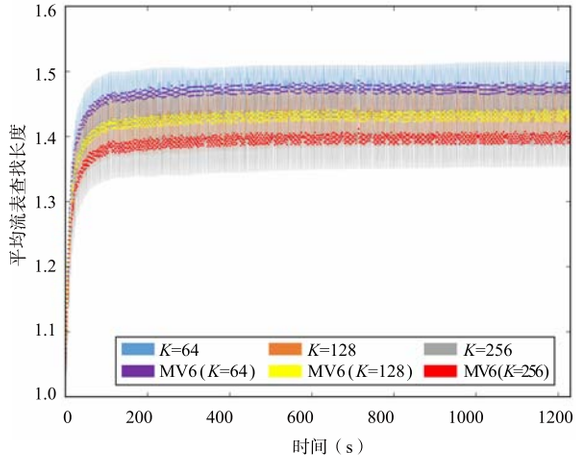

图8　C1场景下MFFTL取不同K值时的表现

场景二至场景三

在C2-C3中,我们分别在流表执行实行时间和流表访问长度两个维度进行了对比实验,为了多个数据集间使用相同的实验参数,窗口K统一设为256.

从图9～12中实验结果可以看出,在短时间刻度和长时间刻度下,FFTL和MFFTL在流表访问时间和查找长度两个维度均有显著下降,FFTL在时间维度下降约14.3%,在查找长度维度下降约16.7%;MFFTL在时间维度下降约为17.1%,在查找长度维度下降约20.2%.

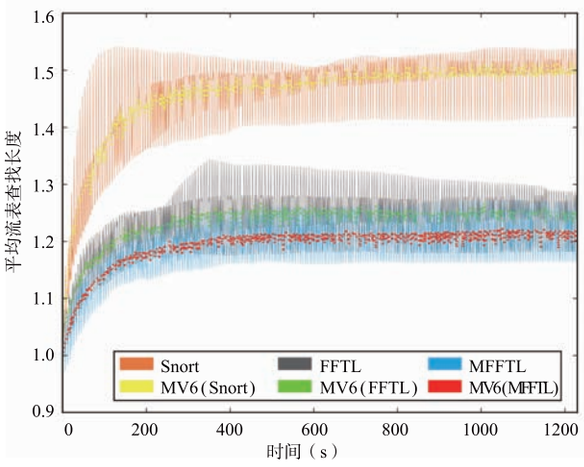

图9 C2场景下查找长度对比

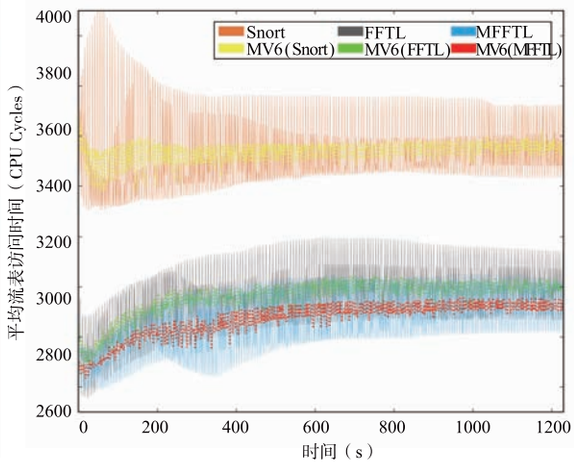

图10 C2场景下流表访问时间对比

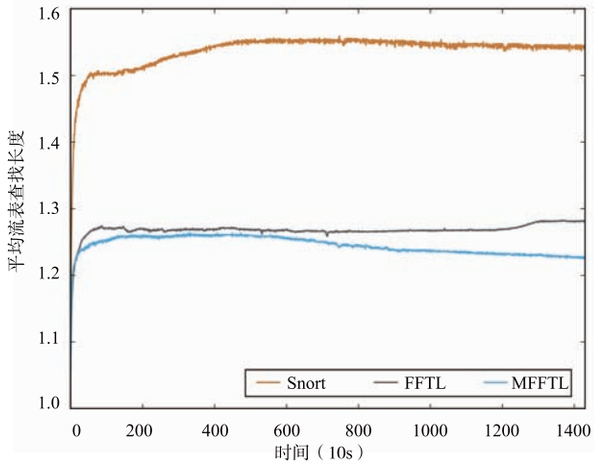

图11 C3场景下查找长度对比

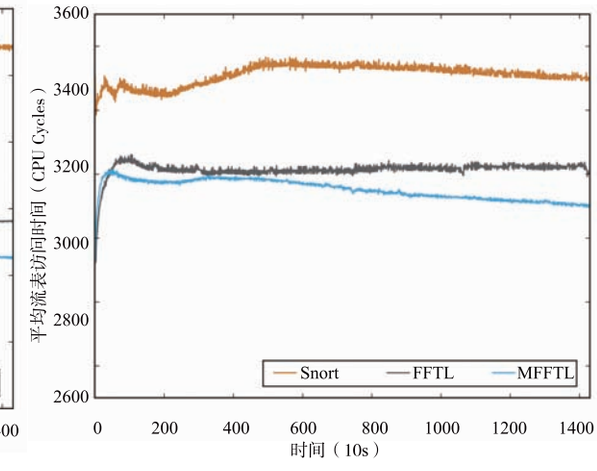

图12 C3场景下流表访问时间对比

6 结论

本文通过对骨干网流量进行测量,分析高速骨干链路的流量局部性特征,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,在此基础上设计实现了一种快速流表查找方法,并对聚合后的调度策略进行改进,引入阈值调度策略和饥饿避免机制.在给出理论复杂度分析的同时,对提出的快速流表查找方法在不同的流量场景下从流表访问时间和流表查找长度两个维度进行了实验评测.实验结果表明,在多种流量环境下,提出的快速流表查找方法均可以提高流表的访问效率.

参考文献

[1] NAM G,PATANKAR P,KESIDIS G,et al. Mass purging of stale tcp flows in per-flow monitoring systems[A]. Proceedings of 18th Internatonal Conference on Computer Communications and Networks[C]. San Francisco:IEEE,2009.1－6.

[2] PAN T,GUO X,ZHANG C,et al. Tracking millions of flows in high speed networks for application identification[A]. Proceedings of 31st Annual IEEE International Conference on Computer Communications[C]. Orlando:IEEE,2012.1647－1655.

[3] NOURELDIEN N A,OSMAN I M. A stateful inspection module architecture[A]. TENCON 2000:Proceedings of Intelligent Systems and Technologies for the New Millennium[C]. Kuala Lumpur:IEEE,2000.259－265.

[4] DHARMAPURIKAR S,SONG H,TURNER J,et al. Fast packet classification using bloom filters[A]. Proceedings of the 2006 ACM/IEEE Symposium on Architecture for

Networking and Communications Systems[C]. California: IEEE,2006:61 – 70.

[5] Snort-an open source network intrusion prevention system [OL]. http://www. snort. org,2013-12-30/ 2014-09-13.

[6] An Implementation of an E-component of Network Intrusion Detection System[OL]. https://sourceforge. net/projects/libnids/files/,2010-03-14/2015-03-02.

[7] CORMEN T H,LEISERSON C E,RIVEST R L,et al. Introduction to Algorithms [M]. Cambridge: MIT Press, 2009. 253 – 285.

[8] NAM G,PATANKAR P,LIM S H,et al. Clock-like flow replacement schemes for resilient flow monitoring[A]. The 29th Int'l Conference on Distributed Computing Systems [C]. Quebec:IEEE,2009. 129 – 136.

[9] Sandvine-Intelligent Broadband Networks [OL]. https://www. sandvine. com/downloads/general/global-internet-phenomena/2014/2h-2014-global-internet-phenomena-report. pdf,2014-11-21/2015-02-26.

[10] 张广兴,等. Internet 城域出口链路流量测量与特征分析[J]. 电子学报,2007,35(11):2092 – 2097.
ZHANG G X,et al. Internet traffic measurement and characteristic analysis on output link of metro area network [J]. Acta Electronica Sinica,2007,35(11):2092 – 2097. (in Chinese)

[11] BONOMI F, MITZENMACHER M, PANIGRAH R, et al. Beyond Bloom filters: om approximate membership checks to approximate state machines[J]. SIGCOMM06, 2006,36(4):315 – 326.

[12] MEINERS C R, LIU A X, TORNG E. TCAM razor: a systematic approach towards minimizing packet classifiers in TCAMs[J]. IEEE/ACM Transactions on Networking, 2010,18(2):490 – 500.

[13] SONG H,DHARMAPURIKAR S,TURNER J,et al. Fast hash table lookup using extended bloom filter: an aid to network processing [J]. ACM SIGCOMM Computer Communication Review,2005,35(4):181 – 192.

[14] AYUSO P N. Netfilter's connection tracking system[J]. Login:the Magazine of USENIX & SAGE,2006,31(3): 34 – 39.

[15] YOON S,KIM B,OH J,et al. High Performance Session State Management Scheme for Stateful Packet Inspection [M]. Berlin: Springer Berlin Heidelberg, 2007. 591 – 594.

[16] 周舟,等. 一种基于并行 Bloom Filter 的高速 URL 查找算法[J]. 电子学报,2015,43(9):1822 – 1840.
ZHOU Z,et al. Fast URL lookup using parallel bloom filters[J]. Acta Electronica Sinica, 2015, 43 (9): 1822 – 1840. (in Chinese)

[17] 谢鲲,等. 布鲁姆过滤器代数运算探讨[J]. 电子学报,2008,36(5):869 – 874.
XIE K,et al. Algebraic operations on bloom filters[J]. Acta Electronica Sinica,2008,36(5):869 – 874. (in Chinese)

[18] 王洪波,等. 高速网络超连接主机检测中的流抽样算法研究[J]. 电子学报,2008,36(4):809 – 818.
WANG H B,et al. On flow sampling for identifying super-connection hosts in high speed networks [J]. Acta Electronica Sinica,2008,36(4):809 – 818. (in Chinese)

[19] LAKSHMINARAYANAN K, RANGARAJAN A, VENKATACHARY S. Algorithms for advanced packet classification with ternary CAMs[J]. Acm Sigcomm Computer Communication Review,2005,35(4):193 – 204.

[20] Center for Applied Internet Data Analysis[OL]. http://www. caida. org/d-ata/passive/passive_2014_dataset. xml,2015-01-20/2015-02-26.

作者简介

王　鹏　男,1990 年生,山东菏泽人,硕士研究生. 主要研究方向为网络安全、高性能网络.

张　良　男,1980 年生,辽宁丹东人,博士. 主要研究方向为计算机系统结构、微处理器设计与验证.

周　舟(通信作者)　男,1983 年生,湖南长沙人,博士. 主要研究方向为网络安全、高性能网络.

E-mail:zhouzhou@ iie. ac. cn

刘庆云　男,1980 年生,河北衡水人,博士. 主要研究方向为网络安全.

方滨兴　男,1960 年生,黑龙江哈尔滨人,博士. 主要研究方向为网络安全.