# 数据中心网络测量方案研究

# 1 背景介绍

在数据中心网络中，网络测量在网络监控中起着非常重要的作用，而一个好的测量方法是提高网络性能的一个重要途径。网络测量解决的具体问题主要有两种：一种是大流(Heavy Hitter)的检测，或者说是top-k流的检测，这个问题要求能够通过某些算法从网络中的众多流中提取出大流进行存储，并能给出这些大流的流量值，而现在针对这种问题的测量的方法一般是基于Counter的测量方法，用有限的Counter统计网络中的大流，经典的基于Counter方法主要有Space Saving，Frequent，RAP等。另一种是流量矩阵估计问题，要求在给定一个流的ID，就能给出这个流的真实大小或者估计值，而这个估计值的误差需要保证在一定的范围中，解决这种问题主要是通过基于Sketch的网络测量方法，通过在数据平面安装Sketch数据结构，将网络流哈希到对应的桶中计数，查询时也是从哈希表中读取相应哈希值的计数值，经典的基于Sketch的方法主要有CM Sketch，C Sketch，CU Sketch等。现有的测量方法为了适应现在高速网络，都在最求常数时间复杂度，因此我们提出的测量方法也要求在常数时间内进行网络测量。但是现有方法还有许多不足之处，例如：不能同时解决Heavy Hitter和流量矩阵估计问题，不能适应复杂网络流量分布多变的情况，不能给出一个确切的误差范围等等。下面详细介绍了Space Saving和CM Sketch算法，并针对他们的不足之处，提出了两种网络测量方法。

# 2基于Counter的测量方法

基于Counter的测量方法，主要是通过有限的计数器存储有限的大流，而小流则是忽略或是只给出一个大致估计值，所以基于Counter的测量方法给出的误差界一般都是确定的。目前基于Counter的测量方法主要有Frequent， Space Saving，RAP等，这些方法的主要思路都是在有限的计数器中只记录大流的数据包，而忽略小流的数据包，只是当计数器的个数达到上限时，处理的方法不一样。这些方法能够保存大流在有限的Counter中,并且采用hash操作能够使得运行时间在常数时间范围中。但是这些方法的缺点在于只能解决Heavy Hitter或top-k问题，对于流量矩阵估计问题解决得不够好，并且由于采用hash操作，不能很好的处理hash冲突而导致运行时间变长。

## 2.1 Space saving算法

Space saving算法能够保证在Counter数足够的情况下，能够保证top-k条流都被记录下来，并且算法能够达到常数时间复杂度，对于Heavy Hitter的检测有非常好的性能。主要思想如下，T保存所有Counter，每次数据包i到达时，若没有空闲的计数器，它那么这个新到达的数据包将替换所有计数器中计数值最小的那一个j，而新的数据包的计数值为j的计数值加1，同时还将替换前的计数值记录到中。具体的算法如下：

**SpaceSaving**(m)

**1** n0;

**2** ;

**3 for each** i **do**

**4** n←n+1;

**5 if** **then**

**6 else if** **then**

**7** ;

**8** ;

**9**

**10 else**

**11**

**12** ;

**13** ;

**14** ;

## 2.2 Space Saving实现方式

通常Space Saving可以采用链表形式的数据结构Stream Summary实现，可以达到常数时间复杂度。Stream Summary将计数值放在一个桶里面，然后每个桶用双向链表按计数值从小到大的顺序连接，记数值相同的counter用链表连接到相应计数值的桶中，每次查找最小的counter只需要在链表头中查找就行了，如图1所示为的Stream Summary实现Space Saving算法的示例：



图2.1 Stream Sumary实现Space Saving

图2.1所示示例最大计数器个数为2，数据流为S={X,Y,Y,Z}，(a)(b)(c)分别为每个数据包到达时Stream Summary的状态。(a)当X和Y到达时，由于计数器个数没有达到上限，所以这两个包都被记录下来，并且指向了value=1的桶中；(b)当又到达一个Y时，由于Y已经记录在链表中，则增加一个value=2的桶，将Y移动到value=2的桶中；(c)当Z到来时，由于没有多余的Counter了，所以将最小的value的桶中的X替换为Z，记为X的计数值，再将这个Counter移动到value=2的桶中，删除value=1的桶。利用这种数据结构可以达到时间复杂度为O(1)，因为每次查找最小计数值时只需要在桶的链表头查找即可。

## 2.3 Space Saving算法分析

显然Space Saving算法需要的空间为O(k)，如果利用Stream Summary数据结构可以达到运行时间为O(1)。对于没有记录在Space Saving的小流可以就用最小的那个Counter值来估计，但是这对于这些小流的估计值的误差是比较大的。

对于任意分布的数据流来说，要想确保检测到误差为松弛的top-k流(即假设表示第k大的流的值，表示流*i*的值，对于任意满足的流*i*，一定记录在Space Saving中)，必须保证Space Saving的Counter数为，其中表示流的个数。

对于常见的Zipf分布的数据流来说，若参数，只需要Counter数至少为，即可以保证准确地解决top-k的流的检测问题；若参数，只需要Counter数至少为，即可以保证准确地解决top-k的流的检测问题。

## 2.4其他方法

除了Space Saving以外，经典的基于Counter的方法还有Frequent，Lossy Counting，RAP等。Frequent当Counter满的时候是通过每个计数器都减1的操作，然后将新流替换计数值为0的计数器，相比Space Saving来说测量误差更大；RAP与Space Saving不同的在于当新的流的数据包到来时，并不是所有的包都替换最小的Counter，而是按照最小计数值的倒数的概率来替换最小Counter，这样可以使得大流的计数值增长得越来越快，小流的计数值增长得越来越慢，这样可以减少大流被小流替换的概率，这样使得RAP需要的Counter数比Space Saving更少。

# 3 基于sketch的测量方法

这一部分介绍基于sketch的测量方法。3.1节以CM sketch为例，详细介绍基于sketch方法的基本流程，3.2节对CM sketch进行误差分析，在此基础上3.3节介绍C sketch与CU sketch的基本原理，3.4节对于sketch测量方法的优缺点进行讨论。

## 3.1 Count Min sketch（CM sketch）的基本原理

Packet

h1

h4

图 3.1 CM sketch的基本结构

CM sketch的基本结构如图 3.1 所示，直观上看，CM sketch数据结构的主体是一个由计数器组成的*d*（图中*d=*4）个长度为*w*（图中*w*=5）的哈希表组成二维数组，并且，这*d*个哈希表所选用的哈希函数彼此无关。

在收到数据报文时，CM sketch首先提取数据报文所对应的流ID。继而，将该ID同时在*d*个哈希表中所对应的计数器值增加该数据报文的权重（例如数据报文的长度）。

由于每个流ID所对应的计数器都至少记录了该流的流量，所以，给定流ID，该ID所对应的任一计数器值一定大于其真实值。因此，在查询流量时，CM sketch在其对应计数器数值中取最小值作为最终的查询值。在下一小节中，将会具体分析这一查询值的误差。

## 3.2 CM sketch的误差范围

将所有流量之和记作。首先考虑单个哈希表的查询误差，对于任意一条流，它在该条哈希表上查询误差的期望为，并且该误差值非负。由马尔科夫不等式，该查询误差大于的概率至多为1/*e*，其中，*e*为自然对数。将*e*/*w*记作ε，对于单个哈希表，我们有：

现在综合考虑全部的*d*个哈希表，在所有哈希表中，查询误差全部大于的概率为,将记作*δ*，即*d*=ln(1/*δ*)，那么对于任意流，我们有：

即，CM sketch至少以1-*δ*的概率，查询误差小于。

## 3.3 C sketch与CU sketch简介

C sketch与CU sketch均是在CM sketch的基础上修改得到的网络测量模型。

C sketch在哈希表的更新策略上进行了更改。对于每一个哈希表，C sketch都额外维护了一个哈希函数gi(·)，该哈希函数的值域为{-1,1}。在进行计数时，C sketch会依照gi(·)的值决定在该计数器上执行相应增加或减少操作。而在查询时，C sketch将gi(·)与对应的查询值相乘，并返回这些查询值的中位数。

CU sketch直接根据CM sketch出现误差的原因进行了改进。CU sketch在进行更新时，仅对到达流ID对应的所有计数器中值最小的进行更新，在保证CM sketch优点的同时，减小了其可能出现的误差，但相对来说维护过程较为复杂。

## 3.4 sketch的优缺点

sketch最大的优点就是它通过哈希存储避免了存储流ID（通常为104比特），解放了存储资源，使得其在相同存储空间下，计数器数量是基于counter的6~8倍。由于counter数量众多，因此sketch可以准确地测量小流的流量。

sketch的缺点主要体现在两方面：由于未存储流ID，因此，sketch只能被动地返回流量的查询值，而没有有效的方法单独维护大流计数，使得sketch无法及时返回异常流量；此外，由于交换机处理速度的限制，sketch的宽度*d*通常为4，无法保证单个流的测量精确程度。

# 4 基于FlowMap的方法

针对Space Saving算法不能够很好地估计小流的流量这一缺点，我们提出了一种基于FlowMap的检测方法。主要思想是先将数据包加入一个Space Saving中，然后将Space Saving替换出来的小流加入到一个FlowMap中，这样大流被留在了Space Saving中，小流被加入到FlowMap中。而小流加入FlowMap的过程主要是通过两级hash操作，第一级hash先将流hash到一个Counter分组中，然后第二级hash决定将它的值加入到哪一个Counter中。在估计时则根据两次hash的映射关系构建出汇聚矩阵A，求解Y=AX来估计流量矩阵X，其中Y是Counter上的值。

## 4.1 FlowMap的主要步骤

如图4.1所示，首先当数据包到达时先进过一个BloomFilter确定这个包是否是已经记录过的流，如果不是，则将其ID记录到流ID表中，如果已经记录过，则直接进入Space Saving中；然后若从Space Saving中有被替换出来的流，则将它的ID和计数值经过第一级hash，选择一个Counter组；再在相应的Counter组中进行第二级hash，分别hash到相应的Counter中计数。

对于每一个counter组，我们可以根据流的映射关系，得到一个方程组AX=Y，其中A就是流的映射矩阵，X为流的大小，Y为counter值的大小。如果A的秩等与流的数目，那么我们可以唯一的确定每个流的大小，但是如果counter数目受限的情况下，我们无法准确的得到流的大小，为此我们可以采用流估计方法，设计优化目标。我们可以将流估计问题建模为如下的优化问题：



显然，上面的优化问题是一个线性规划问题。问题求解的复杂度主要由约束和变量的数目决定。为了降低求解的难度，我们将counter分成了若干的组，这样每组counter只对应少量的约束和变量，而且每组counter记录的流是完全不同的，因此可以并行求解。



图4.1 FlowMap步骤

## 4.2 FlowMap分析

相对于Space Saving来说，FlowMap将小流也进行了统计，而且通过hash操作，在更新的时候可以达到常数时间复杂度。另外由于FlowMap进行了分组，每个Counter分组中保存的流是不同的，因此每个分组都可以构建一个相互无关的线性规划问题，减少了问题的规模，而且可以并行计算。

但是由于Space Saving的误差，有可能会出现大流被替换出来加入到FlowMap中的情况，并且由于BloomFilter也有误差，可能会时记录的流ID比真实情况少，这两种情况就可能使得流量估计误差变大。

## 4.3 FlowMap当前进度及下一步工作

目前正在进行实验仿真评估中，我们从网上下载真实的网络流量数据集进行仿真实验，并以估计流量矩阵的均方误差作为评估数据。在之后的工作中，我们希望能采取一些措施够减少FlowMap的误差，并且希望在NS3平台中进行网络范围的仿真实验。

# 5 sketch与counter混合模型

Sketch与counter两类方法各有优劣，而我的第一部分工作就是尝试将sketch与counter两种测量方式进行混合，力求获得更为全面的测量性能。

# 5.1 基本算法流程

Packet

Space Saving

Sketch

替换流量

图 5.1 sketch与counter的混合模型

混合模型的基本更新流程如图 5.1 所示。在收到报文之后，混合模型首先尝试在Space Saving中进行更新。如果在这一过程中出现了替换操作，则将Space Saving中被替换掉的流ID及其对应的流量记录到sketch中。这样，就保证了所有到达的流量在Space Saving与sketch中记录且只被记录了一次，其算法如下所示。

**Function UPDATE（ID, v） #流ID及其流量**

**1**在SpaceSaving中更新

**2 if** SpaceSaving 中发生了替换：

**3** 在Sketch中更新被替换掉的ID及对应的c-ε

**4 *end if***

**Function QUERY（ID）**

**5*****if*** Space Saving中记录了ID：

**6** ***return***SpaceSaving.query(ID)

**7** **else**:

**8*****return*** sketch.query(ID)

## 5.2 当前工作进展

现阶段已经搭建了一个算法验证平台，并且在UCLA数据集上测试了该混合算法，该测试结果显示在总存储资源消耗相同的情况下，混合算法对于流量测量的精确程度要比单纯的CM sketch与Space Saving都高，这是由于混合模型中的Space Saving截断了流入sketch中的大流流量，提高了其测量的精确性，使其在counter数量不占优的情况下反而比单纯的sketch测量的更准确。

## 5.3 下一步工作内容

接下来的工作将在两个方向上进行：

一方面，下一步将在NS3中仿真实时的交换机环境，测试混合算法的吞吐量，并对其进行改进。

另一方面的工作将集中在将不同的sketch模型与counter模型进行混合，并分别进行算法验证，测试其测量的准确性。