目 录

[1 研究概述 4](#_Toc158121490)

[2 引言 4](#_Toc158121491)

[2.1. 研究背景及意义 4](#_Toc158121494)

[2.2. 背景知识 6](#_Toc158121495)

[2.2.1. 问题定义 6](#_Toc158121496)

[2.2.2. 单流基数估计 6](#_Toc158121497)

[2.2.3. 多流监测 7](#_Toc158121498)

[3 Couper算法 8](#_Toc158121499)

[3.1. 研究动机和概览 8](#_Toc158121501)

[3.2. 基础架构 10](#_Toc158121502)

[3.3. 减小重叠误差 12](#_Toc158121503)

[3.4. 减小哈希冲突所带来的误差 14](#_Toc158121504)

[3.5. 延伸应用：超级传播者检测 16](#_Toc158121505)

[4 数学分析 17](#_Toc158121506)

[4.1. 误报率 17](#_Toc158121508)

[5 实验分析 21](#_Toc158121509)

[5.1. 实验场景 21](#_Toc158121511)

[5.1.1. 参数设置 21](#_Toc158121512)

[5.2. 逐流基数估计 24](#_Toc158121513)

[5.3. 消融分析和参数分析 25](#_Toc158121514)

[5.4. 超级传播者检测 26](#_Toc158121515)

[6 硬件实现 28](#_Toc158121516)

[7 参考文献 31](#_Toc158121517)

插图清单

[图 1 (a) 先前技术采用的数据结构。(b) 真实数据集中的高度偏斜的基数分布（①），当前数据结构在内存效率方面表现不佳（②）。 9](#_Toc158143313)

[图 2 ①“Couper”的基本架构。其中①的(y轴)表示流f到达第二层的概率, 表示为流基数(x轴)的函数。③展示了一个使用“优惠券”估计基数的例子。 11](#_Toc158143314)

[图 3重叠误差问题 13](#_Toc158143315)

[图 4解算的一个简化样例 13](#_Toc158143316)

[图 5解算的一个样例 15](#_Toc158143317)

[图 6细粒度合并的优势 16](#_Toc158143318)

[图 7流大小 () 时的误报率和大流的估算误差. 23](#_Toc158143319)

[图 8逐流基数估计的准确性和空间利用率 27](#_Toc158143320)

[图 9吞吐量 27](#_Toc158143321)

[图 10消融分析和参数分析 28](#_Toc158143322)

[图 11硬件实现的工作流 29](#_Toc158143323)

[图 12 P4实现和CPU实现的准确率对比 31](#_Toc158143324)

附表清单

[表 1单流基数估计器 6](#_Toc158142669)

[表 2 符号含义 11](#_Toc158142670)

[表 3 推荐参数 25](#_Toc158142671)

[表 4数据集的统计数据 25](#_Toc158142672)

[表 5 逐流基数估计的指标 26](#_Toc158142673)

[表 6超级传播者检测的指标 26](#_Toc158142674)

[表 7 ratio error分布的关键分位数 27](#_Toc158142675)

[表 8 超级传播者检查的精确率，召回率和F1-Score 29](#_Toc158142676)

[表 9由Couper带来的准确性提升 30](#_Toc158142677)

[表 10硬件资源利用率 33](#_Toc158142678)

# 研究概述

从高速数据流中估计每个流的基数有许多应用，如异常检测和资源分配。尽管现有的近似算法可以跟踪单流基数，但在监测多流基数的算法方面仍然存在挑战，尤其是在基数分布不平衡的情况下。现有方法采用统一的sketch数据结构，占用大量内存以实现高精度。此外，现有方法难以在线速处理数据流的硬件上部署。

本研究提出了一种内存高效的度量框架Couper，可以在不平衡基数分布下估计多流的基数。本研究提出了一种基于经典优惠券收集器原理的两层结构，其中许多老鼠流被限制在第一层，只有潜在的大象流被允许进入第二层。Couper的双层结构可以更好地适应实践中不平衡的基数分布，并实现更高的存储效率。本项目在软件和硬件上都部署了Couper框架，并在现实数据流和合成数据流下进行了广泛评估。评估显示，与最先进的技术相比，Couper的内存效率提高了20倍以上。

# 引言



## 研究背景及意义

数据流可以被视为多个不相交流的并集，其中可提炼出的统计数据通常可以分为两类：一类统计数据基于流大小估计[7]、[32]、[46]、[45]，其中流中的每个项目仅包含流ID ，并且流中的所有项目对流大小的贡献相等；另一类统计数据基于基数估计 [2], [12], [40], [9], [37]，每个项目都携带一个额外的元素ID ，以区分属于同一流的元素，流中的每个不同元素只计数一次，而不考虑出现的次数。

许多实际应用[28]、[22]、[15]、[16]可以简化为基数估计模型，其中和的定义取决于特定的应用场景。在生产数据中心，为了实现负载平衡，运营商需要及时获得每个服务器维护的并行TCP连接的数量，因此可以将发送到同一服务器的所有数据包视为一个流，并测量每个流的不同源IP地址的数量作为其基数。此外，通过跟踪访问每个文件的不同用户的数量，网络内K-V存储系统[17]可以获取每个内容的流行程度，并相应地调整缓存优先级以提高性能。谷歌的数据分析系统，如Sawzall[28]、Dremel[22]和PowerDrill[15]，通过对所有搜索请求进行建模，将包含同一关键字的请求视作流，用户标识符（例如IP地址）作为每个流中的元素，可以定期估计搜索同一关键字的不同用户的数量。基于安全方面的考量[11]、[23]、[27]、[29]、[36]，数据分析系统应及时检测和报告基数极高的流，此类数据流被称为超级传播者，因为超级传播者可能会发出DDos攻击或恶意端口扫描的信号[31]。

与流大小估计类的统计任务相比，基数估计的中心挑战是需要消除冗余数据。由于存储开销巨大，通过记录所有项目来计算确切的基数是不切实际的。现有的方法通过概率数据结构来近似地估计基数，例如，BJKST[2]、HyperLogLog[12]、[16]、MRB[9]、SMB[37]等，此类数据结构被称为基数估计器。

目前，单流基数的测量已经被广泛研究。使用上述任何一种算法，只需1KB 内存，就可以准确地估计包含数百万个不同项目的流的基数。尽管如此，跟踪多流基数的挑战仍然存在。现有的方法遵循 CM-Sketch [7]的思想: 它们统一地将内存空间划分为由所有流共享的许多单元，并使用基数估计器作为插件。这种统一的布局在实际应用中存储效率较低。具体而言，真实的数据流通常服从高度倾斜的分布，即，流的主要部分是老鼠流(即具有小基数) ，而少数流是大象流(即具有大基数)。在统一布局下，所有单元拥有相同的大小，并且都能够容纳最大的流量。结果，那些只记录老鼠流的单元浪费了大量的内存。

现有的方法不能很好地扩展，因为随着数据流包含更多的流，它们的内存成本急剧增加：当前的大数据流可能在5分钟内包含了数十万个流，导致在DRAM和持久存储中维护其数据结构的成本相当高。此外，为了实现线路速处理，这些测量功能需要在硬件设备上实现部署，包括可编程交换机[3]、[42]、智能网卡[1]和FPGA[38]、[42]。它们小而快速的片上存储器（例如SRAM）强制限制了数据结构的大小。因此，需要设计具有高内存效率的基数测量框架以满足上述需求。

目前现有的一系列提高内存利用率的技术只能在一定程度上缓解内存紧张的问题，因为它们都没有针对不平衡的数据分布进行优化。例如，VHS[41]和CSE[44]提出以更精细的粒度在流之间共享存储器，即，最小存储器共享单元是位(bit)或寄存器，而不是完整的基数估计器。这可以减少内存浪费，但也会增加哈希冲突的概率，一旦平均每流内存大小相对较低，就会大大降低准确性。

本研究提出了Couper来实时执行每流基数估计和超级传播者检测。从本质上讲，为了解决不平衡的分布问题，Couper为老鼠流保留了较少内存，同时为大象流保留了更多的内存。Couper通过设计两层sketch布局来实现这种自适应分配。通过经典优惠券收集器的原理，老鼠流被限制在第一层，每个流只消耗十几个位，从而显著节省内存。本研究通过减轻重叠偏差（分层设计引入的系统误差）和使用细粒度单元组合策略减轻哈希冲突误差来进一步优化精度。

本研究的贡献：

•本研究提出了一个内存高效的框架Couper来测量多流的基数。

•本研究将Couper扩展到检测超级传播者，并将其通用化为元框架，可用于优化现有的超级传播者检测算法。

•本研究对Couper进行了数学分析。

•本研究使用现实数据流和合成数据流来评估其与现有技术的性能对比。结果表明，内存效率得到了提高（例如，降低了16×～20×的内存成本，以实现相同的精度）。

•本研究在软件和硬件（新兴的可编程交换机）中实现了Couper，源代码可在Github[8]上获得。

## 背景知识

### 问题定义

设为一个包含个元素的数据流。中的每一项都包含一对标识符. 具有相同flow ID 的项形成了一个流。元素ID 用来区分同一个流中的项。可以视作多个不相交流的集合，即，。本研究的目标是各级每个流的基数，即每个流中元素的种类数。

### 单流基数估计

现有的单流基数估计解决方案包括LC、MRB和HL。

**Linear Counting（LC）**[40]维护一个位图（bitmap），大小为（初始化为0）。在

更新阶段，对于每一个进入的元素，都通过一个哈希函数并返回一个地址，然后把地址i所在位设为1，即。在查询阶段，我们计算中为0的位的比例（记为V）。最后，对真实传播值 的最大似然估计为：

表1：单流基数估计器

Table

Description automatically generated表 1单流基数估计器

当中只有一个“0”时，LC达到“满”状态，在“满”状态下的估计值称为其容量，即。

**Multi-Resolution-Bitmap（MRB）**[9]采用个-位宽的位图组成并将其排列成层次结构。每个传入项目只能将更新应用于层次结构的一个级别，后继级别相较前驱级别接收更新的可能性以指数级降低。具体而言，将随机哈希函数应用于传入的项目并输出一个位串，其中每个位为0或1的概率为，然后我们计算其中的前导零的数量（用表示）并在第层中记录，因此第层的采样概率为。在查询阶段，MRB将每一层视为一个独特的位图，并将LC应用于每一层。由于位图（LC）的准确性随着其负载因子（“1”位的比例）的增加而降低，MRB仅使用负载因子低于阈值的位图。

**HyperLogLog++（HLL）**[16]估计器由个-位寄存器组成。在更新阶段，对于每一个输入元素，HLL对其进行哈希返回一个比特序列，然后我们令并更新为，其中表示前导零（leading zeros）的个数加1.在查询阶段，HLL返回真实传播值的估计值 为：

其中是修正因子。

### 多流监测

多流监测的一个直接的解决方案是通过哈希表为每个流分配一个单独的基数估计器(例如 HLL) ，当流数量很大时开销可能过高。更重要的是，解决哈希冲突会导致无限制的数据包处理延迟，难以在线处理高速数据流。现有的解决方案在流之间共享内存以降低内存成本，可以分为基于sketch的解决方案和基于采样的解决方案。

基于sketch的解决方案可以视为CM-Sketch的变体。它们将内存空间排列成一系列基数估计器。在处理过程中，每个流都与个单元（使用随机哈希函数）关联以记录其元素。值得注意的是，由于哈希冲突，任何单元都可能被多个流共享，因此单元给出的估计是映射到它的所有流的组合基数，这具有正偏误差。先前的工作采用了不同的策略来降低误差。gSkt[47]独立地进行个估计，并返回最小值；VHS[41]和CSE[44]通过设计虚拟估计器，在更细粒度的流之间共享存储器；rerSkt[38]与Count Sketch[5]有相似的想法，它在主估计器和补估计器之间划分背景流量。通过从主估计器中减去背景流量，它在统计上消除了误差。这种策略基于这样一种假设，即背景流量可以平均分配给每个流量——尽管它对大流量很有效，但对小流量来说变化很大。

基于采样的解决方案[34]通过非重复采样（Non-Duplicate Sampling, NDS）选择不同的元素。NDS维护一个大位图来处理所有流的重复，每个数据包以采样概率映射到位图中的空位，并且无论数据包是否被采样，该位都将被设置为“1”。每个采样分组将被发送到哈希表，该哈希表跟踪每个流的采样分组。然而，哈希表可能过大而无法在快速但紧凑的片上存储器中完全实现NDS。为了执行在线处理，[34]提出将哈希表卸载到片外存储器，然而受到片上和片外吞吐量比的限制, 整体采样率相当低。而且它需要在片上和片外之间占用相当大的带宽，这些带宽应该留给更重要的应用。由于在实践中实现基于采样的解决方案成本高昂且复杂，本研究重点关注基于sketch的解决方案。

# Couper算法



## 研究动机和概览

Diagram

Description automatically generated

图 1 (a) 先前技术采用的数据结构。(b) 真实数据集中的高度偏斜的基数分布（①），当前数据结构在内存效率方面表现不佳（②）。

**现有框架及其不足之处**

现有的针对多流基数测量的方法借鉴了Count-Min Sketch的内存共享思想。它们将一块内存排列成一个基数估计器数组，如图1（a）所示。每个流随机选择k（我们图中的两个）个估计器来记录它的元素（⟨f, e⟩对）。由于哈希冲突，一个估计器可能输出多个流的组合基数。一方面，需要大量的估计器来避免流之间的严重哈希冲突；另一方面，大部分内存在只记录“小流”的估计器中保持未使用状态，而“小流”在实际数据分布中占大多数。因此，相当数量的估计器将遭受低利用率的问题。

为了具体说明我们的关注点，我们在图1(b)中展示了三个真实世界数据追踪的分布，即MAWI、CAIDA、FACEBOOK。请注意，在所有3个数据追踪中，基数大于15的流量少于20％，而最大基数可达到105。此外，我们在图1(a)中使用sketch，通过实验研究了处理这些数据集时的内存利用情况，其中我们尝试了MRB和HLL作为基数估计器，并且保证哈希碰撞率最多为10％。图1(b)表明，在所有（估计器，数据集）组合下，超过90％的估计器都具有利用率（即修改的位或计数器的比例）低于20％。可以预测的是，如果我们使用LC [40]、MinCount [2]或其他内存利用效率较低的估计器，这种未充分利用的情况可能会更加严重。

**优惠券收集者原理及其应用**

我们首先回顾优惠券收集者问题。玩家希望从一系列礼品盒中收集优惠券。总共有b种不同的优惠券，每个盒子随机包含一个1到b之间的正整数优惠券，直到打开盒子才能看到。为了收集t种不同的优惠券，预期所需的盒子数量为券收集者原理可以帮助我们将众多的小流从数据流中分离出来，并以内存高效的方式处理它们。

具体来说，当一个数据项出现时，我们将一个编号为的“优惠券”分配给流f，其中是一个随机哈希函数，⊕表示拼接操作。由于的结果被假设为均匀分布，故b种优惠券将以相同的概率被收集。因此，流f包含的不同元素越多，它获得的不同优惠券就越多。如果我们为赢得收集设置一个阈值，那么基数大于和小于 的流可以被大致分离。

在实际应用中，b张优惠券的成员关系可以用长度为b的位图表示，其中第i位的值“1”（“0”）表明编号为i的优惠券已经（没有）被收集。因此，我们可以为每个流分配一个位图来跟踪它所收集的优惠券。一旦一个流f的“1”位数量超过阈值，我们就可以得出结论，f不是一个小流，并将其（剩余的）元素记录在一个草图中（例如，图1(a)）。这种方法的主要优势是，即使是一个只有几十位的小位图就足以估测一个小流，我们因此可以减少由大量小流造成的内存浪费。而且，这些小流被阻止进入sketch，其哈希碰撞率显著降低。

Diagram

Description automatically generated

图 2 ①“Couper”的基本架构。其中①的(y轴)表示流f到达第二层的概率, 表示为流基数(x轴)的函数。③展示了一个使用“优惠券”估计基数的例子。

## 基础架构

如图2所示，Couper采用2层架构。第一层由个位图组成，每个位图占用b位，第二层由个基数估计器组成，每个基数估计器占用位。我们将第一层占用的内存比例记为。有两种可选的估计器——HLL++和MRB——因为它们能够高效地记录大流量（为方便起见，我们后面将HLL++简称为HLL）。取决于每个估计器所需的标准误差。所有的位和计数器在开始时都初始化为0。为方便起见，关键符号在表II中总结。

Table

Description automatically generated表 2 符号含义

**插入 (算法 1)：**

在接收到项目后，我们首先转向并通过一个统一的散列函数 定位流f的位图。所选的位图，即 ，用于记录流收集的优惠券。中“1”比特的数量 — 用表示 — 决定了接下来的操作：

Case 1: . 这意味着流f还没有收集到足够的优惠券。因此，我们通过将TB[h(f)]中相应的比特设为“1”，为流f分配一个编号为的优惠券。

Case 2: . 这意味着流f已经收集了足够的优惠券，或者不幸地与其他也映射到的流发生冲突，它们共同完成了收集。在这个时候，我们转向第二层，并保持不变。在第二层，我们用估计器来记录，其中是一个随机哈希函数。

我们在图2中展示了一个例子(②)。一个有4个不同元素的小流被限制在Layer1中，因为它只获得了3个不同的优惠券，小于 的阈值。而另一个流在项目发生时,收集了7个优惠券中的4个，那么的剩余元素就到达了Layer2。

**估计（算法2）：**

估计某个流量的基数时，我们首先转向第一层（Layer1）并通过遍历位图来检查流量f收集到的优惠券数量（即）。与插入阶段类似，估算方案也取决于：

Case 1: . 这意味着流f没有到达第二层（Layer2），它的所有元素都用于收集优惠券。因此，我们可以根据它收集的优惠券数量来估计流f的基数。根据Linear Counting算法[40]，流f基数的最大似然估计（MLE）应该是。

Case 2: . 这意味着流f在过去某个时刻到达了第二层（Layer2），它的一些元素被记录下来。我们可以使用和分别估计第一层（Layer1）和第二层（Layer2）记录的元素的基数。最后，要估计，一个简单的解决方案是将这两个估计值相加。这可能会导致重叠误差——尽管在大多数情况下可以忽略不计——我们将在下一节讨论如何减轻这种误差。

## 减小重叠误差

本章节介绍原始的估计方案如何导致了重叠误差，以及如何改进来减小重叠误差。

双层结构的设计可以提高内存利用率，但会导致有些元素被重复计数，违背了基数的定义。两层估计器会重复地记录同一个元素，图3具体地说明了这个现象：流在中记录前6个元素（这个集合记为），收集到了4个奖券，随后剩余的元素（记为）都被记录在中。算法2实际上估计的是，但真实的基数是。因此算法2有估计偏差。

Diagram, schematic

Description automatically generated

图 3重叠误差问题

一个减小误差的方法是估计并将它减去。因此当容易被估计时，初步的改进如图4所示：和分别由两个位图和记录（这两个集合有两个相同的元素和）。因为和长度均为7，并且使用相同的哈希函数，这两个位图可以被按位与，得到的位图就可以被用于估计。

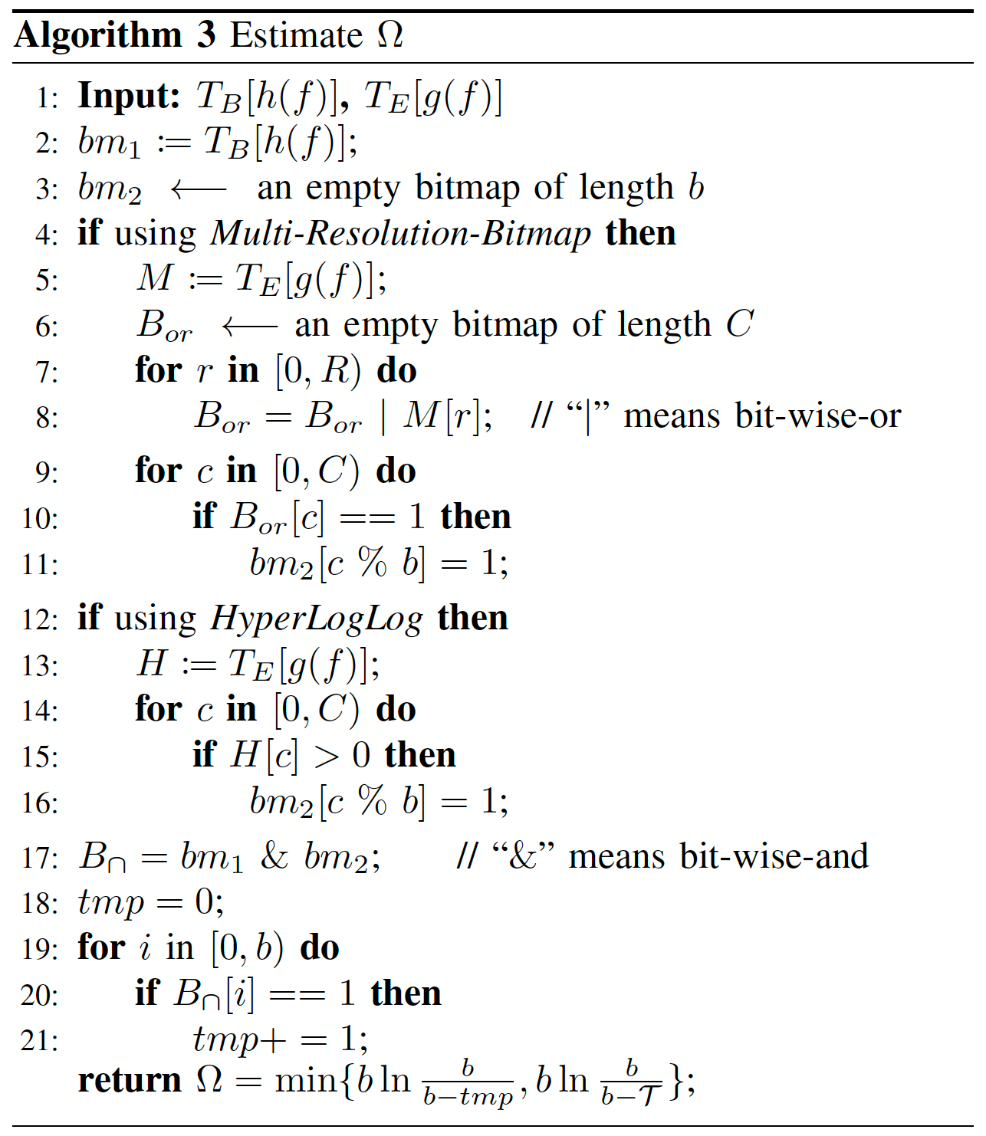
Text

Description automatically generated

图 4解算的一个简化样例

在双层设计中，就是，但是估计器不是一个和等长度的位图。朴素的想法是为每个中的估计器附加一个同样长度的位图，并用相同的哈希函数执行同样的操作。这个方法很简单，但会带来额外的存储开销。于是这个附加的仅在查询阶段才被创建。

为了得知第一层位图的每一位与第二层估计器的哪一个计数器产生重叠误差，就需要设计特定的哈希函数让他们二者联系起来。在原始版本中，可以是任一哈希函数，而现在这样定义：。如表一所述，MRB和HLL使用来决定哪一列来记录元素，所以MRB和HLL的第列和位图中的第列产生了重叠误差，算法3就能估计重叠误差，图5展示了这个流程。以MRB为例，当一个元素被记录在MRB的第列时，那么在第一层记录它的位图会将第位修改为1。因此在查询阶段，找出MRB中的非空列，它们的列号组成的集合为，构造这样的位图，它与按位与，就能得到重叠误差。



算法3

Diagram

Description automatically generated

图 5解算的一个样例

最终，算法2的第十行被修改为

## 减小哈希冲突所带来的误差

在3.2节中，每个流在每层只会被映射到单个位图或者估计器，如果有两个流被映射到同一个单元，即产生哈希冲突，就会产生比较大的误差。因此记录时应该将每个流映射到多个单元中，查询时将各单元估计的数组合，得到最终的估计。

3.4.1 用多个单元共同记录

每个流在第一层会使用个哈希函数映射到对应的位图，在第二层会使用个哈希函数映射到对应的估计器，这些单元都会执行各自的记录操作。查询阶段按如下所述，合并这些计数单元。

3.4.2 合并计数单元

gSkt[47]也使用多个单元共同记录，但是它直接将多个单元估计值的最小值作为结果。这是一种粗粒度的方法，并没有充分利用计数单元特定的数据结构。一方面，相同类型的计数单元使用相同的哈希函数（例如位图共用函数，MRB共用函数），所以不同的计数单元对单个流的更新所执行的操作是一样的；另一方面，与流产生哈希冲突的更新不太可能将这些计数单元中本该置0的位同时“污染”为1（例如，的第位由于哈希冲突为1，但的第位可能仍然为0）。因此，我们可以比较多个计数单元，找到它们不同的位，从而得到更准确的结果。综上所述，细粒度合并（FGM）如算法4所示。

Text

Description automatically generated

算法4

图6举例说明了为什么FGM效果比gSkt更好。gSkt的查询结果是三个位图的最小值，而Couper使用FGM，将被污染位的数量减小到了1，因而比gSkt的误差小5。

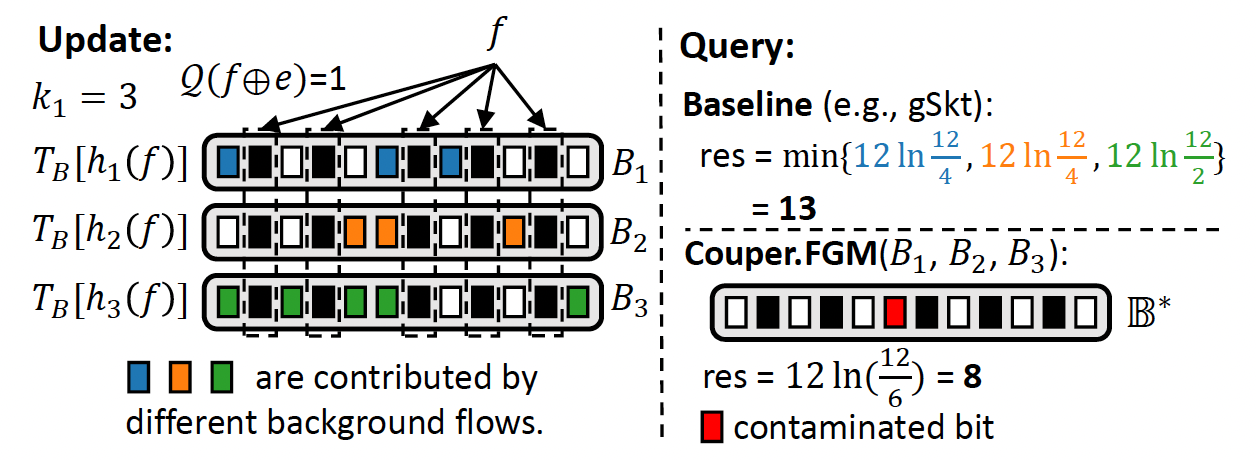


图 6细粒度合并的优势

FGM的输出是单个计数单元，所以可以直接被应用到之前的设计中。具体而言，算法2和算法3中的和被替换为FGM和FGM。

## 延伸应用：超级传播者检测

**用Couper检测超级传播者**

超级传播者是指基数超过一个预先设定的阈值的流。Couper可以在完成基数大小测量的同时完成超级传播者检测。

记录超级传播者需要频繁比较不同流的基数，这种比较应该被高效执行来保持高吞吐量。首先，双层结构将小流都限制在第一层，所以只需要将达到第二层的流作为超级传播候选者。其次，为了提高效率，Couper使用更简便的比较方法，不是精确地算出流的基数,而是计算“概要”来代替。具体地，使用HLL时，概要是所有寄存器之和，即；使用MRB时，概要是行数最高的非空行，即。因为概要和真实基数正相关，所以可以通过比较概要来实现超级传播者检测。

用概要代替的优势不仅是计算更方便，还在于每次更新时都便于修改。以HLL为例，每当HLL记录一个新元素并发生了更新操作时，它的概要也会被增加。比如HLL的一个寄存器由2变成5，与此同时，概要就会被增加3。每个估计器的概要仅占用几个比特，而赋予了Couper完成超级传播者检测任务的能力。

长度为的桶表被用于记录超级传播候选者，每个桶包含一个流ID和它的概要。当流更新了一个概要时，它和它的概要就会被汇报到桶表。为了缓解哈希冲突，它会被两个哈希函数各映射到一个桶中，桶负责记录概要更大的流ID。桶表的插入流程如算法5所示。

Text, letter

Description automatically generated

算法5

在查询阶段，用算法2估计桶表中所有流的基数大小，那些基数超过阈值的流被报告为超级传播者。

**用Couper的思想提升现有检测框架**

目前的超级传播者检测方法如SS、VBF和CDS需要记录所有的流和元素对，因此它们很有可能让小流冒充超级传播候选者。但Couper的第一层可以限制住大多数小流，不让它们进入第二层的估计，所以可以减小它们冒充超级传播候选者的可能。此外，一些算法如VBF和CDS可能会产生从未出现过的流ID当成超级传播者，而Couper使用算法2重新估计，避免了这个问题。

# 数学分析

我们针对 和 的 Couper 进行正式分析，这是我们实验中的默认配置。



## 误报率

我们研究了小流量（mice flow）被错误报告给第二层的概率。严格来讲，我们将基数不大于 的流量定义为小流量，因为它们在设计上应该只收集 张优惠券。为了方便起见，我们在本节中将所有其他流量称为大流量（elephant flows）。

**引理1** 假设 表示具有 个不同元素的老鼠流获得 张优惠券的概率。那么

由于空间限制，证明在这里省略。 设 和 是流 的两个关联的位图（bitmap）。我们将分别考虑以下3种情况下的误报率。

: 和 都不和大流冲突.

: 只有 和 其中之一和大流冲突.

: 和 同时和大流冲突.

给定大象流的数量为 ，一个位图不被任何大象流映射到的概率为 。那么我们有, 和 .

**引理2** 一个收集了q张优惠券（仅靠自己）的小流将以低于以下表达式的概率被误报：

其中

N1 为所有小流的基数总和，

***证明：***

*1)* 令 表示f收集的优惠券。在 () 中设置为“1”的位可能由（1） 和（2）其他老鼠流的噪声贡献。我们可以将小流提供的噪声视为独立的，因此 () 中的每一个bit 有 的概率接受噪声。其中 表示所有小流的基数之和。在FGM（细粒度合并） 之后，产生的位图 中的第 个 bit () 被污染，当且仅当 and 的第 个 bit 都收到了噪声, 这种情况发生的概率为 . 所以 中被污染的bit 个数（记为 ）应当服从分布 . 当 时，在 情况下 时会发生误报，其概率为

*2)* . 不失一般性，我们假设 与大流冲突了. 我们悲观的假设 , 即 中的每个bit 都有 的概率接收到噪声。 所以在 情况下 时发生误报的概率与 相似：

*3)* . 因为这种情况发生的概率很低，我们直接悲观的假设在情况 下流 会被误报：

根据全概率定律，我们得到以下定理。

**定理1** 具有X个不同元素的老鼠流将以一个受限的概率被误报：

对于那些被限制在第一层的流, 误差始终低于 , 这是一个非常小的值. 所以我们只分析那些到达第二层的大象流。 令 为流 中被第二层记录的元素。我们最终对该流的基数估计可以写成

其中 = Query(FGM()) ，它是对 = 的估算. 我们首先分析 的分布。

近似服从正太分布，且它的期望和方差如下：

其中

由于空间限制，我们将只讨论HyperLogLog 的情况：

令 是流 关联的两个 HLL. 考虑任意计数器 , 以及它的对应部分 , 我们将 记录的互不相同的元素分别记为 , 它包含*A)* 流 的元素，记为 ，和 *B)* 背景流贡献的噪声，分别记为 。注意到背景流包含所有被第二层记录的元素（记为 ）排除掉流 本身的元素。为方便起见，我们将所有背景元素视为独立分布，于是得到：

经过FGM 后， 和 中较小的那个会贡献给最终的HyperLogLog 。因为 , 和 , 正相关。 记录的噪声将会是 (也就是说，我们假设小的计数器值对应于小的噪声). 两个独立同分布的高斯分布变量的较小值也可以被建模为一个高斯分布的变量。然后我们有

令 为计数器 记录的噪音，则 是相互独立的， 记录的总噪声将应当是

记录的总元素 为 。因为HLL 是近似无偏的，它的标准差是 , 我们有

其中 。因为 且 , 可以被视为正太分布，期望为 ，方差为 .

最后，我们得到 是正太分布的，它的期望和方差分别如下：

**定理2** 对于某大流 的基数估计，记为 ，有如下的置信区间：

其中 是置信等级（confidence level）， 是高斯分布的 分位数（例， 当 ）。

***证明：***

回顾我们之前的结论 , 因为 , 我们有

因为 的置信区间为 , 结论成立。

为了验证我们分析的准确性，我们将理论和实验结果在MAWI数据集上进行比较。Couper使用默认参数 ()，占用空间大小为1.5MB。下图显示，理论误报率与实验误报率非常接近。下图（b）和（c）分别显示了Couper使用HLL和MRB时大象流的估计准确性。橙色和蓝色线表示置信区间的上限和下限（α = 95%），图中的每个绿色点代表一个流。

Chart, line chart

Description automatically generated

图 7流大小 () 时的误报率和大流的估算误差.

# 实验分析



## 实验场景

### 参数设置

1. ：sketch算法中2,3,4是常见的参数选项。在实验中我们设置。
2. ：HLL或MRB是E的递减函数: 。我们期望它小于一个阈值，因此我们设置。
3. ：如果位图中有太多比特被设置为 1，就会产生很大的估计误差，因此我们设置 来避免位图饱和。
4. ：这两个参数是Couper中最重要的参数，用于表征每一层有多少个单元以及在给定分布的情况下有多少流量会到达第二层，并且本质上确定了两层的哈希碰撞率（记作）。因此我们应该让越小越好，这里我们定义。

应该依赖于当前数量分布D，内存预算M和参数（）。D形式上定义为一个1维数组，用于表述具有基数x的流的数目。第i层的流只有当它的所有元素都和其他流共享的时候才会发生哈希碰撞。因此碰撞率可以表述为。在该公式中，由决定：。表示为第i层记录的流的数量，可以大致按照如下计算：。其中标识误报率，可以使用前文中的公式1在给定参数（）的估计。因此也可以表述为（）相关的函数（）。我们可以在参数空间（）中搜索最优的参数组合来最好的适配当前的：。

Couper与每个数据集的ground-truth分布无关，因此在实验中，我们通过特定的分布以及先验的选择来生成D。这里我们选择的是分布[30]，该分布可以通过指定来设定偏斜程度，它被广泛的用于模拟不平衡数据流的分布[32],[43],[18],[46]。我们设置到上面的等式中（该设置标识每一个流平均分配10Bytes）。在下表中显示了随着偏斜程度的变化情况，的取值范围魏1.0-1.5，这是因为现实世界的数据流分布通常不会低于Zipf-1.0，很少超过Zipf-1.5[20]。根据该表，我们设置（）的取值范围是([20,8],[0.5,0.65])。举例而言，如果我们根据历史数据先验的知道当前数据的偏斜程度接近Zipf-1.1，那么我们应该设置。

出于比较的公平性，我们对于所有数据集设置的偏斜程度为1.3，在下列实验中使用，并且不对参数进行微调。关于的分析会在下面讨论。

Table

Description automatically generated表 3 推荐参数

**数据集**

我们使用了三个真实场景的数据集和一个合成数据集:

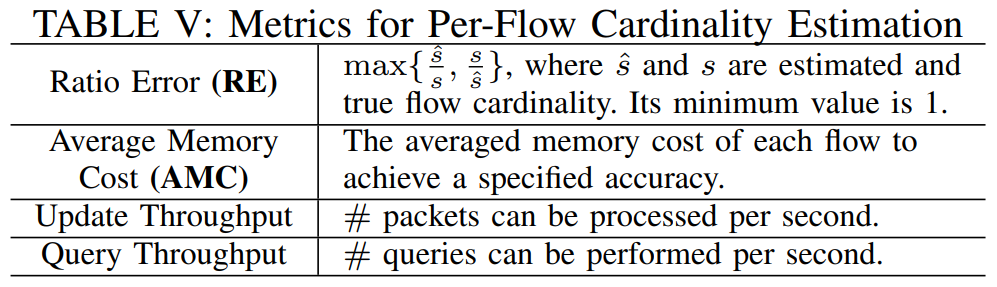
1. CAIDA19：这是 2019 年从 CAIDA [4] 收集的匿名 IP 跟踪流。我们把目标IP作为流ID把源IP作为元素ID。
2. MAWI：这是一个交通数据 WIDE 项目[21]的 MAWI 工作组维护的交通数据存储库。流ID和元素ID定义为每一个包里面的源IP和目的IP
3. FACEBOOK：我们使用在2017年手机的FACEBOOK页面数据集[10]。文件中的每一行都是一条边 有两个节点。节点代表网页，边代表网页之间的相互关联。我们用每条边的第一个节点 作为流 ID，第二个节点作为元素 ID。
4. Zipf：我们生成了数据集，流的基数分布符合Zipf-1.0分布。

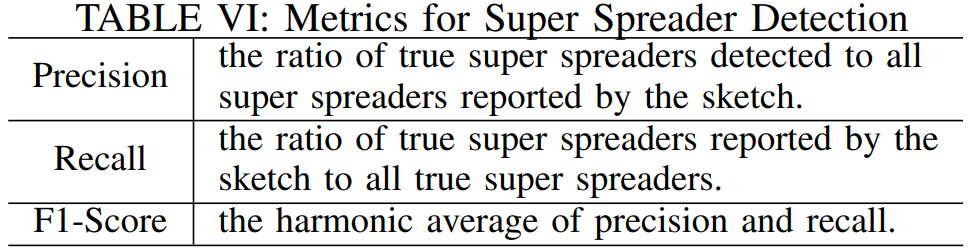
我们将每个数据集分为几个子集，在每个时间窗口中对其中一个子集进行评估，然后得出所有 epoch 的平均结果。结果。我们在表 IV 中总结了不同 数据子集的统计数据汇总于表 IV。

Table

Description automatically generated表 4数据集的统计数据

**指标**：见表格5 和 6

表 5 逐流基数估计的指标

表 6超级传播者检测的指标

**实现：**我们使用C++实现了Couper和其他的框架。所有实验在一个服务器上，该服务器有一对8C16T CPU（Intel Xeon Silver 4215R CPU @ 3.20GHz）和总共192G的DDR4 RAM，系统为Ubuntu18.04.1。我们使用128比特的MurmurHash3 [24]作为默认的哈希哈数。在数据集上运行每个实验之前，我们会将整个数据集加载到内存中，以避免磁盘 I/O 开销。

## 逐流基数估计

参与比较的基线工作包括gSkt[47],rerSkt[38]和VHS[41]。对于gSkt和rerSkt我们使用了两种类型的估计器。这些对比实验的参数都保持它们原有论文中的设置不变。

**内存有效性和准确性**

首先我们设置了一个预期平均比率误差（expected Average Ratio Error），并且在所有框架中比较了AMC（图8（a）-（b））。具体而言，我们给每一个框架设定了初始内存占用知道达到预期精度。结果表明为了预期平均比率误差1.5和1.25，Couper的AMC起码比最佳的原方案低了和。我们在不同的内存分配上分析了平均比率误差。结果显示，在内存占用为2MB的情况下，只有Couper可以保证在所有的数据集的平均比率误差都小于1.5。图中没有rerSkt-LC和gSkt的原因是因为它们的误差太大。另外我们在表7中汇总了80百分位误差和99百分位误差的数据。在99百分位误差中，Couper-MRB比rerSkt-HLL, gSkt-HLL和VHS低了。

Table

Description automatically generated表 7 ratio error分布的关键分位数

Graphical user interface

Description automatically generated with medium confidence

图 8逐流基数估计的准确性和空间利用率

**吞吐量**

图9比较了所有框架更新和查询的吞吐量。所有框架的更新吞吐量都很高，但是所有基线方案的查询吞吐量只能达到更新吞吐量的0.1%~2%。这是意料之中的，因为每一次估算都需要成百上千次的内存访问和加法运算。相比之下，Couper拥有更高的查询吞吐率，原因是因为大多数小流只会在第一层记录，而估算他们的事件复杂度是O(b)。b是一个相对较小的数。由于小流占据大多数，Couper事故宣布可淮海的均摊查询吞吐量。

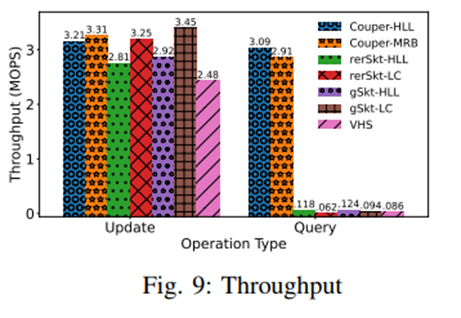


图 9吞吐量

## 消融分析和参数分析

**消融分析：**这里我们主要区分两种优化技术，即消除重叠误差（ROE）和细粒度合并（FGM）。

**ROE：**图10（a）比较了启用和禁用ROE时的详细的误差分布情况。流根据他们的真实基数被放置于不同的桶中。我们在图中绘制了每一个桶的平均系数误差，误差条用于标识0.1-0.9的百分位误差。我们观察到基数在[12-26]区间的流可以通过ROE显著提高精度。当流的基数超过30的时候，提升开始变得不明显。这是符合预期的，因为重合误差的范围是，因此ROE对于基数远大于的流只有边际效应。

FGM：现在我们分析我们的FGM策略相对于传统的格融合策略的提升，我们称格融合策略为基线策略。基线策略和FGM有以下不同：1) 当检查流量是否收集足够的coupon时，基线策略会计算在选定位图中的1的数量并且当他们不小于的时候返回真。2) 当接收到一个查询的时候，基线策略会对每一层的哈希单元进行估算，并且把各层的最小结果相加作为最终估算结果。我们比较了基线和FGM方案的误报流的数量和平均系数误差，如图10(b)所示。FGM有更低的误报率并且减少了大约40%的平均误差。最后图10(c)比较了每种方法带来的准确率提升。 图10如下：

Diagram

Description automatically generated

图 10消融分析和参数分析

**的参数敏感性分析：**

默认情况下（）可以在（[20, 8]、[0.5, 0.65]）范围内选择。我们在之前的实验中使用了 (b = 12; µ = 0.6)。现在，我们将研究不同参数组合的性能（以AMC作为指标）。我们选取表3中不同的偏度以及对应的（）进行了实验。结果在图10(d)中。结果表明不同的数据集倾向于不同的参数，这意味着他们有不同的偏移程度。例如FACEBOOK相对于MAWI和CAIDA有着更小的偏移程度。我们同样发现，次优的参数会影响Couper的表现，但是仍优于当下最先进的方案。

## 超级传播者检测

对比的基线方案包括Vector Bloom Filter (VBF) [19], Spread Sketch (SS) [35] 和 Connection Degree Sketch (CDS) [39]。由于Spread Sketch (SS) [35]的作者没有指定SS中的参数，我们将它设置为0.1（这是尝试了0.05，0.1，0.2之后的最优结果）。在CDS中我们使用了两个reversible sketch和一个validation sketch。我们为Couper设置Z=2048.

**准确性：**表8评估了所有框架在内存占用为0.5MB和1MB时在 CAIDA 和 MAWI 跟踪数据上的精度、召回率和F1-score。超级传播者的阈值设置为，这里，是所有流基数的加和。我们观察到在所有情况下，VBF和CDS都有极低的准确率及召回率，他们的F1-score都在0左右。我们同样观察到最先进的方法SS在MAWI上表现优异，但是在CAIDA数据上准确率很低，特别是在内存比较少的时候。我们的方案Couper达到了最高的准确率和F1-score，同时保持了很高的召回率。

Couper的高准确率受益于它的两层设计，由于第一层可以过滤掉大部分小鼠流量，因此只有一小部分流量被视为超级流量并且汇报给，这让Couper可以轻松处理大量流量。相比之下，其他基线方案必须记录所有流的元素。在内存紧张的情况下，大量小鼠流会被记录在同一个桶中，这让超级传播者占有的桶和大量小鼠流占有的桶很难分辨，最后造成严重的精度损失。

Table

Description automatically generated with medium confidence表 8 超级传播者检查的精确率，召回率和F1-Score

使用Couper提高前人的工作：我们使用Couper的第一层来优化前人的超级传播者检测算法，结果如表9所示。出于公平比较，Couper不会导致额外的内存占用，因此我们只给Couper的第一层分配了200KB资源。表9中的结果显示了所有的算法都通过Couper获得了准确性和F1-Score的提升。

Table

Description automatically generated表 9由Couper带来的准确性提升

# 硬件实现

Diagram

Description automatically generated

图 11硬件实现的工作流

以上讨论都假设使用CPU软件来实现Couper算法，本节讨论如何将其实现在可编程硬件上。尽管在软件实现中，主机上的CPU可以灵活地支持任何类型的操作，但CPU以及网

卡都可能成为性能瓶颈：目前的高端商用网卡只能达到200Gbps~400Gbps的吞吐量；CPU单核的吞吐量则更低，即使在高度优化的系统配置下也难以达到100Gbps。作为对比，使用PISA(Protocol independent switch architecture)架构的Tofino P4可编程交换机提供了多个100Gbps/400Gbps端口，总吞吐量可以达到10Tbps量级，远高于主机上的高端商用网卡。然而，PISA架构也对算法设计施加了更多的限制。下面我们简要介绍了如何将原始版本的Couper改编为硬件版本，使其可以在Tofino 1芯片上运行。

图11展示了Tofino中PISA架构的逻辑视图，并给出了硬件版Couper的算法流程。Tofino在流水线(Pipeline)中处理数据包，Pipeline按数据包的入向(Ingress)、出向(Egress)分成两部分，之间由Traffic Management(TM)模块相连。Pipeline的两个部分几乎完全一致，都由解析器(Parser)、匹配-动作单元(Match-Action Unit, MAU)、反解析器(Deparser)构成，MAU共有12级(stage)，包含了Couper的核心处理逻辑。12级MAU使用前馈(Feed-forward)的方式处理数据包，即数据包只能从一个stage传递到下一个stage。这样的架构中存在几个关键硬件限制。C1) 所有的内存资源都是stage的本地资源，当处理一个数据包时，不能访问分配给以前阶stage的寄存器。C2) 一个stage内的所有操作都是并行执行的。例如，给定两个操作和，如果取决于的结果，则它们必须放在不同的阶段。另外，一个寄存器在其所属stage只能被访问一次。C3) 不允许有循环依赖关系。这可以从C1和C2中推导出来。C4) ALU和控制流支持受限。与C/C++相比，Tofino中只允许一小部分算术和逻辑运算。不支持的操作包括浮点运算、变量间的位运算、循环操作等。

为了使Couper满足硬件上严格的资源约束，我们的修改如下。

1. 修改算法设计和参数设置。将位图大小从12修改至16，因为P4只支持字节对齐的内存访问。将Murmur哈希替换为CRC哈希，因为前者无法在硬件上实现。移除中的Power-of-two设计，因为这会导致循环依赖。
2. 对于硬件不支持操作，使用查表变通。因为C4约束，部分操作在P4中无法直接实现，例如a) 检查位图是否已满b) 更新HLL时计算前导0的个数等。我们使用等价的查表操作来代替。例如，当位图中的个位中，有个置1时，我们认为位图已满，当两个参数分别取12、16时，一个满的位图只可能有种不同的模式。通过将所有模式编码进TCAM查找表，我们能够在1个时钟周期内检查位图是否已满。HLL更新的设计采用类似思路。
3. 为了消除循环依赖并使算法能重复访问内存，重新设计数据包的处理路径。图11展示了算法的完整流程。每个入向数据包首先进入L1检查其所属的流是否获得足够的coupon。未获得足够coupon时，数据包的处理路径为：1)设置P4 resubmit标志，当数据包在ingress反解析器上处理完成后，再次将其注入ingress解析器；2)更新Layer1；3)照常转发数据包。获得足够coupon时，数据包的处理路径为：A)更新Layer2；B)如果Layer2的值改变了，则向egress流水线发送一个摘要，并将其存储在的摘要部分中；C)更新的flowID部分；D)照常转发数据包。

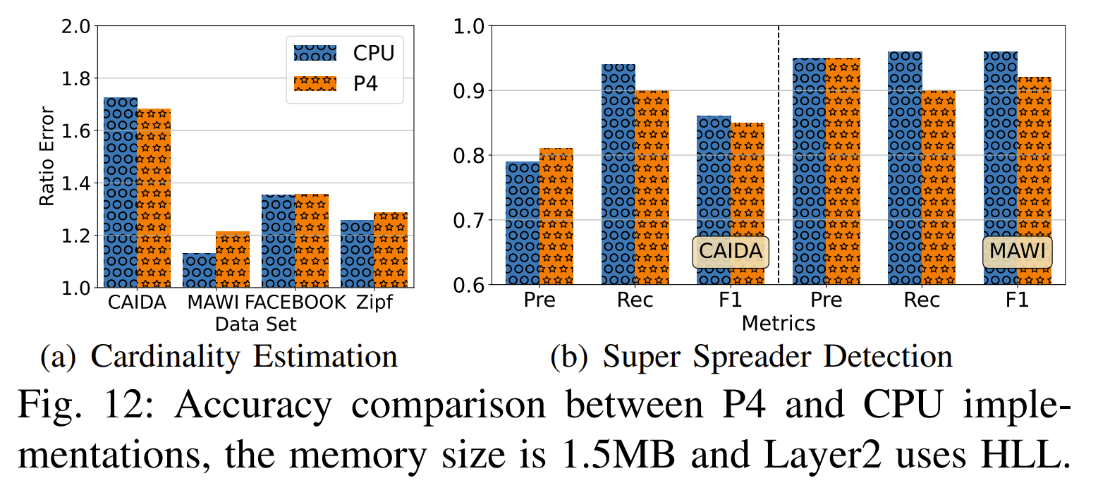
硬件版Couper使用P4\_16实现，表2展示了算法在Tofino1上的资源占用率，图12展示了硬件版算法相较于原版的准确率变化。资源占用方面，算法消耗了22%的哈希分布单元，这是因为Tofino的数组索引操作需要大量Hash操作，而其他硬件资源占用都较低，我们认为Couper能与其他数据平面的算法无冲突的共存。算法准确率方面，A)逐流基数估计中，硬件版算法准确率与原版类似；B)超点检测任务中，硬件版算法的召回率、F1-score略低，主要原因是硬件上并未实现power-of-two choices设计。

图 12 P4实现和CPU实现的准确率对比

表 10硬件资源利用率

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SRAM | TCAM | Instructions | Gateway | Hash Dist Unit |
| 12.6% | 0.35% | 4.43% | 8.33% | 22.22% |

参考文献

1. Agarwal, Anup, Zaoxing Liu, and Srinivasan Seshan. “{HeteroSketch}: Coordinating Network-Wide Monitoring in Heterogeneous and Dynamic Networks.” In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 719–41, 2022.
2. Bar-Yossef, Ziv, TS Jayram, Ravi Kumar, D Sivakumar, and Luca Trevisan. “Counting Distinct Elements in a Data Stream.” In *International Workshop on Randomization and Approximation Techniques in Computer Science*, 1–10. Springer, 2002.
3. Ben-Basat, Ran, Gil Einziger, Roy Friedman, and Yaron Kassner. “Heavy Hitters in Streams and Sliding Windows.” In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, 1–9. IEEE, 2016.
4. Bosshart, Pat, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, et al. “P4: Programming Protocol-Independent Packet Processors.” *ACM SIGCOMM Computer Communication Review* 44, no. 3 (2014): 87–95.
5. CAIDA, 2019. <https://www.caida.org/catalog/datasets/overview/>.
6. Charikar, Moses, Kevin Chen, and Martin Farach-Colton. “Finding Frequent Items in Data Streams.” In *International Colloquium on Automata, Languages, and Programming*, 693–703. Springer, 2002.
7. Chazelle, Bernard, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. “The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Tables.” In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 30–39. Citeseer, 2004.
8. Cormode, Graham, and S Muthukrishnan. “Space Efficient Mining of Multigraph Streams.” In *Proceedings of the Twenty-Fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 271–82, 2005.
9. Cormode, Graham, and Shan Muthukrishnan. “An Improved Data Stream Summary: The Count-Min Sketch and Its Applications.” *Journal of Algorithms* 55, no. 1 (2005): 58–75.
10. Couper, 2022. <https://github.com/CouperProj/Couper.git>.
11. Demaine, Erik D, Alejandro López-Ortiz, and J Ian Munro. “Frequency Estimation of Internet Packet Streams with Limited Space.” In *European Symposium on Algorithms*, 348–60. Springer, 2002.
12. Du, Yang, He Huang, Yu-E Sun, Shigang Chen, and Guoju Gao. “Self-Adaptive Sampling for Network Traffic Measurement.” In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, 1–10. IEEE, 2021.
13. Durand, Marianne, and Philippe Flajolet. “Loglog Counting of Large Cardinalities.” In *European Symposium on Algorithms*, 605–17. Springer, 2003.
14. Estan, Cristian, and George Varghese. “New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice.” *ACM Transactions on Computer Systems (TOCS)* 21, no. 3 (2003): 270–313.
15. Estan, Cristian, George Varghese, and Mike Fisk. “Bitmap Algorithms for Counting Active Flows on High Speed Links.” In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, 153–66, 2003.
16. FACEBOOK, 2017. <https://snap.stanford.edu/data/facebook-large-page-page-network.html>.
17. Fayaz, Seyed K, Yoshiaki Tobioka, Vyas Sekar, and Michael Bailey. “Bohatei: Flexible and Elastic DDoS Defense.” In *24th USENIX Security Symposium (USENIX Security 15)*, 817–32, 2015.
18. Flajolet, Philippe, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. “Hyperloglog: The Analysis of a near-Optimal Cardinality Estimation Algorithm.” In *Discrete Mathematics and Theoretical Computer Science*, 137–56. Discrete Mathematics and Theoretical Computer Science, 2007.
19. Frazier, Peter I. “A Tutorial on Bayesian Optimization.” *arXiv Preprint arXiv:1807.02811*, 2018.
20. Ganguly, Sumit, Minos Garofalakis, Rajeev Rastogi, and Krishan Sabnani. “Streaming Algorithms for Robust, Real-Time Detection of Ddos Attacks.” In *27th International Conference on Distributed Computing Systems (ICDCS’07)*, 4–4. IEEE, 2007.
21. Garnett, Roman. *Bayesian Optimization*. Cambridge University Press, 2023.
22. Hall, Alexander, Olaf Bachmann, Robert Büssow, Silviu Gănceanu, and Marc Nunkesser. “Processing a Trillion Cells per Mouse Click.” *arXiv Preprint arXiv:1208.0225*, 2012.
23. Heule, Stefan, Marc Nunkesser, and Alexander Hall. “Hyperloglog in Practice: Algorithmic Engineering of a State of the Art Cardinality Estimation Algorithm.” In *Proceedings of the 16th International Conference on Extending Database Technology*, 683–92, 2013.
24. Jasny, Matthias, Lasse Thostrup, Tobias Ziegler, and Carsten Binnig. “P4DB - The Case for In-Network OLTP.” *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*, 2022, 1375–89.
25. ———. “P4DB–The Case for In-Network OLTP (Extended Technical Report).” *arXiv Preprint arXiv:2206.00623*, 2022.
26. Jin, Xin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. “Netcache: Balancing Key-Value Stores with Fast in-Network Caching.” In *Proceedings of the 26th Symposium on Operating Systems Principles*, 121–36, 2017.
27. Lerner, Alberto, Rana Hussein, Philippe Cudre-Mauroux, and U eXascale Infolab. “The Case for Network Accelerated Query Processing.” In *CIDR*, 2019.
28. Li, Jizhou, Zikun Li, Yifei Xu, Shiqi Jiang, Tong Yang, Bin Cui, Yafei Dai, and Gong Zhang. “Wavingsketch: An Unbiased and Generic Sketch for Finding Top-k Items in Data Streams.” In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1574–84, 2020.
29. Li, Junru, Youyou Lu, Yiming Zhang, Qing Wang, Zhuo Cheng, Keji Huang, and Jiwu Shu. “SwitchTx: Scalable in-Network Coordination for Distributed Transaction Processing.” *Proceedings of the VLDB Endowment* 15, no. 11 (2022): 2881–94.
30. Li, Tao, Shigang Chen, and Yibei Ling. “Per-Flow Traffic Measurement through Randomized Counter Sharing.” *IEEE/ACM Transactions on Networking* 20, no. 5 (2012): 1622–34.
31. Liu, Weijiang, Wenyu Qu, Jian Gong, and Keqiu Li. “Detection of Superpoints Using a Vector Bloom Filter.” *IEEE Transactions on Information Forensics and Security* 11, no. 3 (2015): 514–27.
32. Liu, Weijiang, Wenyu Qu, Gong Jian, and Li Keqiu. “A Novel Data Streaming Method Detecting Superpoints.” In *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 1042–47. IEEE, 2011.
33. Liu, Zaoxing, Zhihao Bai, Zhenming Liu, Xiaozhou Li, Changhoon Kim, Vladimir Braverman, Xin Jin, and Ion Stoica. “{DistCache}: Provable Load Balancing for {Large-Scale} Storage Systems with Distributed Caching.” In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, 143–57, 2019.
34. Mahajan, Ratul, Steven M Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. “Controlling High Bandwidth Aggregates in the Network.” *ACM SIGCOMM Computer Communication Review* 32, no. 3 (2002): 62–73.
35. Manerikar, Nishad, and Themis Palpanas. “Frequent Items in Streaming Data: An Experimental Evaluation of the State-of-the-Art.” *Data & Knowledge Engineering* 68, no. 4 (2009): 415–30.
36. MAWI, 2019. <https://mawi.wide.ad.jp/mawi/>.
37. Melnik, Sergey, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. “Dremel: Interactive Analysis of Web-Scale Datasets.” *Proceedings of the VLDB Endowment* 3, no. 1–2 (2010): 330–39.
38. Metwally, Ahmed, Divyakant Agrawal, and Amr El Abbadi. “Efficient Computation of Frequent and Top-k Elements in Data Streams.” In *International Conference on Database Theory*, 398–412. Springer, 2005.
39. Miao, Rui, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. “Silkroad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching Asics.” In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 15–28, 2017.
40. Mitzenmacher, Michael, and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge university press, 2017.
41. Moore, David, Colleen Shannon, Douglas J Brown, Geoffrey M Voelker, and Stefan Savage. “Inferring Internet Denial-of-Service Activity.” *ACM Transactions on Computer Systems (TOCS)* 24, no. 2 (2006): 115–39.
42. murmurhash, 2010. <https://sites.google.com/site/murmurhash/>.
43. Nadarajah, Saralees, and Samuel Kotz. “Exact Distribution of the Max/Min of Two Gaussian Random Variables.” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 16, no. 2 (2008): 210–12.
44. Pagh, Rasmus, and Flemming Friche Rodler. “Cuckoo Hashing.” *Journal of Algorithms* 51, no. 2 (2004): 122–44.
45. Park, Kihong, and Heejo Lee. “On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets.” *ACM SIGCOMM Computer Communication Review* 31, no. 4 (2001): 15–26.
46. Pike, Rob, Sean Dorward, Robert Griesemer, and Sean Quinlan. “Interpreting the Data: Parallel Analysis with Sawzall.” *Scientific Programming* 13, no. 4 (2005): 277–98.
47. Plonka, David. “FlowScan: A Network Traffic Flow Reporting and Visualization Tool.” In *LISA*, 305–17, 2000.
48. Powers, David MW. “Applications and Explanations of Zipf’s Law.” In *New Methods in Language Processing and Computational Natural Language Learning*, 1998.
49. Roesch, Martin and others. “Snort: Lightweight Intrusion Detection for Networks.” In *Lisa*, 99:229–38, 1999.
50. Roy, Pratanu, Arijit Khan, and Gustavo Alonso. “Augmented Sketch: Faster and More Accurate Stream Processing.” In *Proceedings of the 2016 International Conference on Management of Data*, 1449–63, 2016.
51. Sapio, Amedeo, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan RK Ports, and Peter Richtárik. “Scaling Distributed Machine Learning with In-Network Aggregation.” *arXiv Preprint arXiv:1903.06701*, 2019.
52. Seijas-Macías, Antonio, and Amílcar Oliveira. “An Approach to Distribution of the Product of Two Normal Variables.” *Discussiones Mathematicae Probability and Statistics* 32, no. 1–2 (2012): 87–99.
53. Song, Cha Hwan, Pravein Govindan Kannan, Bryan Kian Hsiang Low, and Mun Choon Chan. “FCM-Sketch: Generic Network Measurements with Data Plane Support.” In *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, 78–92, 2020.
54. Sun, Yu-E, He Huang, Chaoyi Ma, Shigang Chen, Yang Du, and Qingjun Xiao. “Online Spread Estimation with Non-Duplicate Sampling.” In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, 2440–48. IEEE, 2020.
55. Tang, Lu, Qun Huang, and Patrick PC Lee. “SpreadSketch: Toward Invertible and Network-Wide Detection of Superspreaders.” In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, 1608–17. IEEE, 2020.
56. Ting, Daniel. “Data Sketches for Disaggregated Subset Sum and Frequent Item Estimation.” In *Proceedings of the 2018 International Conference on Management of Data*, 1129–40, 2018.
57. Tirmazi, Muhammad, Ran Ben Basat, Jiaqi Gao, and Minlan Yu. “Cheetah: Accelerating Database Queries with Switch Pruning.” In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2407–22, 2020.
58. Traces, KAGGLE, 2017. <https://www.kaggle.com/jsrojas/ip-network-traffic-flows-labeled-with-87-apps>.
59. Venkataraman, Shobha, Dawn Song, Phillip B Gibbons, and Avrim Blum. “New Streaming Algorithms for Fast Detection of Superspreaders.” Carnegie-Mellon Univ Pittsburgh Pa School Of Computer Science, 2004.
60. Wang, Haibo, Chaoyi Ma, Shigang Chen, and Yuanda Wang. “Online Cardinality Estimation by Self-Morphing Bitmaps.” In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 1–13. IEEE, 2022.
61. Wang, Haibo, Chaoyi Ma, Olufemi O Odegbile, Shigang Chen, and Jih-Kwon Peir. “Randomized Error Removal for Online Spread Estimation in Data Streaming.” *Proceedings of the VLDB Endowment* 14, no. 6 (2021): 1040–52.
62. Wang, Pinghui, Xiaohong Guan, Tao Qin, and Qiuzhen Huang. “A Data Streaming Method for Monitoring Host Connection Degrees of High-Speed Links.” *IEEE Transactions on Information Forensics and Security* 6, no. 3 (2011): 1086–98.
63. Whang, Kyu-Young, Brad T Vander-Zanden, and Howard M Taylor. “A Linear-Time Probabilistic Counting Algorithm for Database Applications.” *ACM Transactions on Database Systems (TODS)* 15, no. 2 (1990): 208–29.
64. Xiao, Qingjun, Shigang Chen, Min Chen, and Yibei Ling. “Hyper-Compact Virtual Estimators for Big Network Data Based on Register Sharing.” In *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 417–28, 2015.
65. Yang, Tong, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. “Elastic Sketch: Adaptive and Fast Network-Wide Measurements.” In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 561–75, 2018.
66. Yang, Tong, Haowei Zhang, Jinyang Li, Junzhi Gong, Steve Uhlig, Shigang Chen, and Xiaoming Li. “HeavyKeeper: An Accurate Algorithm for Finding Top- k Elephant Flows.” *IEEE/ACM Transactions on Networking* 27, no. 5 (2019): 1845–58.
67. Yang, Tong, Yang Zhou, Hao Jin, Shigang Chen, and Xiaoming Li. “Pyramid Sketch: A Sketch Framework for Frequency Estimation of Data Streams.” *Proceedings of the VLDB Endowment* 10, no. 11 (2017): 1442–53.
68. Yoon, MyungKeun, Tao Li, Shigang Chen, and J-K Peir. “Fit a Spread Estimator in Small Memory.” In *IEEE INFOCOM 2009*, 504–12. IEEE, 2009.
69. Zhao, Bohan, Xiang Li, Boyu Tian, Zhiyu Mei, and Wenfei Wu. “DHS: Adaptive Memory Layout Organization of Sketch Slots for Fast and Accurate Data Stream Processing.” In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2285–93, 2021.
70. Zhou, Yang, Tong Yang, Jie Jiang, Bin Cui, Minlan Yu, Xiaoming Li, and Steve Uhlig. “Cold Filter: A Meta-Framework for Faster and More Accurate Stream Processing.” In *Proceedings of the 2018 International Conference on Management of Data*, 741–56, 2018.
71. Zhou, You, Youlin Zhang, Chaoyi Ma, Shigang Chen, and Olufemi O Odegbile. “Generalized Sketch Families for Network Traffic Measurement.” *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3, no. 3 (2019): 1–34.
72. Zhou, You, Yian Zhou, Min Chen, and Shigang Chen. “Persistent Spread Measurement for Big Network Data Based on Register Intersection.” *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1, no. 1 (2017): 1–29.