软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software [doi: 10.13328/j.cnki.jos.007090]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

RobustSketch: 支持网络流量抖动的大流弹性识别方法*

熊 兵[1,2], 刘永青[1], 夏卓群[1], 赵宝康[3], 张 锦[1]

[1](长沙理工大学 计算机与通信学院, 湖南 长沙 410114)

[2](天普大学 计算机与信息科学系, 费城 19122)

[3](国防科技大学 计算机学院, 湖南 长沙 410073)

通信作者: 赵宝康, bkzhao@nudt.edu.cn; 张锦, jzhang@csust.edu.cn

摘 要: 大流识别是网络测量中的一项关键基础性工作, 目前主流的方法是采用概要型数据结构 Sketch 快速统计网络流量, 进而高效筛选大流. 然而, 当网络流量发生抖动时, 大量分组的急速涌入将导致大流识别精度显著下降. 对此, 提出一种支持流量抖动的网络大流弹性识别方法 RobustSketch. 所提方法首先设计基于 Sketch 循环链的可伸缩小流过滤器, 根据实时分组到达速率适应性扩增与缩减其中的 Sketch 数量, 以始终完整记录当前时间周期内所有到达的网络分组, 从而确保网络流量抖动出现时仍能精确过滤小流. 然后设计基于动态分段哈希的可拓展大流记录表, 根据小流过滤器筛选后的候选大流数量适应性增加与减少分段, 以完整记录所有候选大流, 并保持较高的存储空间利用率. 进一步, 通过理论分析给出了所提小流过滤器和大流记录表的误差界限. 最后, 借助真实网络流量样本, 对所提大流识别方法 RobustSketch 进行实验评估. 实验结果表明: 所提方法的大流识别精确率明显高于现有方法, 即使在网络流量抖动时仍能稳定保持在 99% 以上, 而平均相对误差减少了 2.7 倍以上, 有效提升了大流识别的精确性和鲁棒性.

关键词: 网络流量抖动; 大流弹性识别; Sketch 循环链; 可伸缩小流过滤器; 可拓展大流记录表

中图法分类号: TP393

中文引用格式: 熊兵, 刘永青, 夏卓群, 赵宝康, 张锦. RobustSketch: 支持网络流量抖动的大流弹性识别方法. 软件学报. http://www.jos.org.cn/1000-9825/7090.htm

英文引用格式: Xiong B, Liu YQ, Xia ZQ, Zhao BK, ZHANG J. RobustSketch: Elastic Method for Elephant Flow Identification Supporting Network Traffic Jitters. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7090.htm

RobustSketch: Elastic Method for Elephant Flow Identification Supporting Network Traffic Jitters

XIONG Bing[1,2], LIU Yong-Qing[1], XIA Zhuo-Qun[1], ZHAO Bao-Kang[3], ZHANG Jin[1]

[1](School of Computer and Communication Engineering, Changsha University of Science & Technology, Changsha 410114, China)

[2](Department of Computer and Information Sciences, Temple University, Philadelphia 19122, USA)

[3](College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China)

Abstract: Elephant flow identification is a fundamental task in network measurements. Currently, the mainstream methods generally employ sketch data structure Sketch to quickly count network traffic and efficiently find elephant flows. However, the rapid influx of numerous packets will significantly decrease the identification accuracy of elephant flows under network traffic jitters. To this end, this study proposes an elastic identification method for elephant flows supporting network traffic jitters, which is named RobustSketch. This method first designs a stretchable mice flow filter based on the cyclic Sketch chain, and adaptively increases and reduces the number of

* 基金项目: 国家自然科学基金 (62272062, U22B2005, 61972412); 湖南省自然科学基金 (2023JJ30053, 2021JJ30456); 湖南省教育厅资助科研项目 (22A0232, 22B0300); 国防科技计划 (2021-KJWPDL-17); 中央军委装发预研项目 (31511010402); 湖南省研究生科研创新资助项目 (CX20230913)
收稿时间: 2023-05-19; 修改时间: 2023-08-09; 采用时间: 2023-11-10; jos 在线出版时间: 2024-03-26

Sketch in real-time packet arrival rates. As a result, it always completely records all arrived packets within the current period to ensure accurate mice flow filtering even under network traffic jitters. Subsequently, this study designs a scalable elephant flow record table based on dynamic segmented hashing, which adaptively increases and reduces segments according to the number of candidate elephant flows filtered out by the mice flow filter. Finally, this can fully record all candidate elephant flows and keep high storage space utilization. Furthermore, the error bounds of the proposed mice flow filter and elephant flow recording table are provided by theoretical analysis. Finally, experimental evaluation is conducted on the proposed elephant flow identification method RobustSketch with real network traffic samples. Experimental results indicate that the identification accuracy of elephant flows of the proposed method is significantly higher than that of the existing methods, and can stably keep high accuracy of over 99% even under network traffic jitters. Meanwhile, its average relative error is reduced by more than 2.7 times, which enhances the accuracy and robustness of elephant flow identification.

Key words: network traffic jitters; elastic elephant flow identification; cyclic sketch chain; stretchable mice flow filters; scalable elephant flow record tables

1 引 言

网络测量是对网络状态和行为特征进行分析, 进而量化网络各项指标, 在网络管理、网络运维、网络计费、网络安全和流量工程等领域中有重要作用[1]. 随着网络流速的提高, 使用有限的内存准确地进行网络测量越来越具有挑战性. 因此, 基于 Sketch 的数据流处理技术被广泛应用在网络测量领域, 它可以一次性处理数据并立即给出近似估计值. 基于 Sketch 的典型网络测量任务包括: 大流识别[2]、流频率估计[3]、流基数估计[4]、分位数估计[5]以及熵估计等[6]. 针对网络管理的多样化需求, 也有部分研究者开始将 Sketch 结构用于超级主机识别[7]、持续性估计[8]、包延迟估计[9]等任务, 或用于测量新定义的数据流模式如稳定流[10]、突发流[11]等. 其中, 大流识别通常定义为寻找网络流量中数据分组数量大于指定阈值的流, 对大流的精准测量可为异常检测、拥塞控制、网络行为监控和故障分析等应用提供不可或缺的信息[12,13].

目前, 基于 Sketch 的大流识别方法通过哈希函数将所有流映射到紧凑的二维计数器进行统计, 进而通过查询找出大流, 不仅实现了与高速网络相匹配的处理速度, 同时具有较高的测量精度. 然而, 大量小流造成的哈希冲突容易导致测量结果偏大. 此外, 当网络流量发生剧烈波动现象时, 现有大流识别方案的计数器会发生溢出失效, 导致大流的估计值与实际值出现严重偏差, 难以适应网络流量的抖动性. 更为甚者, 当出现网络拥塞、DDoS 攻击等情况时, 可用带宽、分组速率和流大小分布等网络流量特性随之剧变, 而现有大流识别方案的容量固定且需提前设定, 不能根据流量规模变化而变化实现自适应测量, 可能导致部分数据丢失, 显著降低测量性能. 因此, 迫切需要一种弹性识别方法, 实现流量抖动时的大流自适应精确测量.

目前不少研究人员提出预先过滤小流的方法, 以降低大量小流对测量结果的影响, 从而提高大流识别的准确性, 如 Cold Filter[14] 和 Augmented Sketch[15] 等. 然而, 当网络流量抖动尤其是激增时, 分组数量急剧增多, 计数器很快溢出, 导致小流过滤效果显著下降. 针对流量抖动的问题, 已有研究工作优化计数器结构使其能表示更大范围, 以防止计数器溢出, 如 Pyramid Sketch[16] 和 SALSA[17] 等. 然而, 扩大计数器范围将牺牲可用于计数的比特或占用额外的辅助比特. 当网络流速不高时, 还会浪费计数器的高位比特, 降低了内存使用效率. 此外, 也有方案根据已存储的大流数量成倍扩大或压缩记录表, 以适应流量的抖动, 如 Elastic Sketch[18] 等. 然而, 成倍扩展的方式无法灵活应对流量的变化, 可能导致记录表大小过大, 浪费大量存储空间.

针对上述问题, 本文提出一种支持流量抖动的网络大流弹性识别方法. 本文首先分析流量抖动导致现有大流识别方法精度低的原因, 进而采用先过滤小流后记录大流的思想, 设计一种基于 Sketch 循环链的可伸缩小流过滤器和一种基于动态分段哈希的可拓展大流记录表. 然后, 从理论上推导了小流过滤器和大流记录表的误差界限. 最后利用真实网络流量样本, 对本文所提支持流量抖动的网络大流弹性识别方法的性能进行实验验证. 本文的主要贡献总结如下.

(1) 基于小流过滤思想, 提出一种支持流量抖动的网络大流弹性识别方法 RobustSketch, 通过精简的小流过滤器预先剔除大部分小流, 有效降低小流分组对大流记录表产生的哈希冲突, 显著提升大流识别精度的同时, 有效降低了大流记录表的内存开销.

(2) 针对网络流量激增时过多小流产生大量哈希冲突造成过滤器失效的问题, 设计一种基于 Sketch 循环链的可伸缩小流过滤器, 通过适应性扩增与缩减循环链中的 Sketch 数量, 完整记录固定时间周期内的网络分组, 并通过加权平均统计策略更新用于过滤判定的计数器, 保证了小流过滤器的持续有效性.

(3) 针对流量抖动时大流记录表因容量固定容易造成部分大流遗漏的问题, 设计一种基于动态分段哈希的可拓展大流记录表, 可根据大流数量动态分配记录表容量, 并在传入流候选位置已满时, 通过概率衰减策略剔除候选位置中的最小流, 以尽可能保存所有真实大流, 进一步提高大流识别精度.

(4) 给出了支持流量抖动的网络大流弹性识别方法的举例说明, 并从理论上推导了对应的所提方法的误差界限, 最后借助真实网络流量样本验证了其性能的优越性.

本文第 2 节介绍了相关工作. 第 3 节描述了 RobustSketch 的设计思想, 并分别介绍了小流过滤器和大流记录表的结构设计和具体操作. 第 4 节利用真实网络流量样本评估了本文所提大流弹性识别方法的性能. 最后总结全文.

2 相关工作

目前主流的大流识别方法主要是采用紧凑型数据结构 Sketch, 通过哈希映射将流的分组数量统计到 Sketch 的每个计数器向量中, 然后将流映射的最小计数器作为其大小估计值, 进而判定大流, 但无法直接报告出其流标识符[19]. 对此, Tang 等人[20]提出的 MV-Sketch 为每个计数器增加由五元组字段构成的流标识符, 以便报告大流信息. 然而, 由于大量小流分组的影响, MV-Sketch 的流大小估计容易出现较大误差, 导致大流识别精度不高. 随后, Yang 等人[21]设计的 HeavyKeeper 采用指数概率衰减策略驱逐小流, 以减小流大小的估计误差, 但难以驱除中等大小的流, 导致新到达的大流无法存入. 同时, Li 等人[22]提出的 WavingSketch 为每个分组依据其流标识符生成+1 或–1, 使大量小流的分组映射到计数器时相互抵消, 但也可能出现多条大流相互抵消的情况. 进一步, Ye 等人[23]设计的 UA-Sketch 根据连续到达的大流分组数量生成替换概率, 作为发生哈希冲突时判断传入流是否替换已存储流的依据, 有效避免大流被小流替换的情况, 但其在多个映射位置存储同一条大流的流标识符, 导致存储空间的浪费.

针对大量小流对大流识别的干扰, 已有研究工作通过预先过滤小流或筛选大流提高大流识别的准确性. Estan 等人[24]最早提出小流过滤思想, 设计多阶段并行过滤器预先筛除大量小流, 然后精确识别大流, 以减少存储开销. 进一步, Zhou 等人[14]采用层层递进的方式逐级过滤小流, 以提高小流过滤的准确性. 然而, 当网络流量发生剧烈波动现象时, 大量计数器将发生溢出, 进而造成小流过滤器逐步趋于失效. Li 等人[25]设计的 Ladder-Filter 则使用多个 LRU 队列丢弃过滤器中的不常用项, 显著提升过滤效率, 但其在过滤阶段记录流标识符导致存储开销较大. 同时, Roy 等人[15]提出的 Augmented Sketch 采用 CM Sketch 筛选大流, 然后利用高速缓存快速统计大流的分组数量, 以提高整体的吞吐率. Yang 等人[26]提出的 HeavyGuardian 则将 Sketch 划分为轻重两部分, 重部分记录大流, 轻部分从剩余流中选出大流. 但 CM sketch 和 HeavyGuardian 轻部分因哈希冲突导致流大小估计偏大, 容易产生大流误判问题. 此外, 上述方法不能根据流量规模变化实现自适应测量, 难以适应网络流量的抖动性.

针对流量抖动时测量精度显著下降的问题, 已有研究通过优化计数器结构, 以防止计数器溢出. Yang 等人[16]提出的 Pyramid Sketch 设计一种多层金字塔形计数器结构, 每层计数器溢出后将增加到与之相关联的下一层计数器. 然而, 所有层的计数器需要预先分配, 导致内存利用率较低. 对此, Basat 等人[17]提出的 SALSA 将溢出的计数器和它相邻的计数器合并来动态调整计数器的大小, 并通过字节对齐来优化性能, 但需要额外的计算来找出合并过的计数器, 导致吞吐量下降. 同时, 部分研究者通过动态扩展计数器数量, 以在流量抖动时完整记录大流信息. Huang 等人[27]提出的 LD-Sketch 通过动态更改映射桶中关联数组的长度来提供可伸缩性测量. 然而 LD-Sketch 允许任意未存储的流插入到桶中, 大量小流的插入将导致创建大量的计数器进而导致存储开销过大. 进一步, Yang 等人[18]提出的 Elastic Sketch 在用重轻两部分区分存储大流和小流的基础上, 在重部分已满时对其进行复制并合并成原来大小的两倍. 然而, 翻倍后的重部分计数器数量可能远大于大流数量, 导致严重的内存浪费.

3 RobustSketch 方法

3.1 设计思想

在网络流量抖动时, 现有基于 Sketch 的大流识别方法因哈希冲突导致流大小估计值偏大, 进而导致大流识别精度低. 对此, 本文首先采用小流过滤思想, 先把大部分小流剔除, 再将大流进行记录, 以有效降低哈希冲突, 提高大流识别精准性. 进一步, 考虑到网络流量抖动时过滤器的计数器会大量溢出的情况, 本文设计一种基于 Sketch 循环链的可伸缩小流过滤器, 通过适应性伸缩策略保证记录数据的完整性, 然后通过加权平均策略提升过滤器的准确性, 从而实现持续精确过滤小流. 在此基础上, 为保证网络流量抖动时精确记录大流, 本文设计一种基于动态分段哈希的可拓展大流记录表, 在大流数量激增时适应性增加分段, 以尽可能记录所有大流, 同时在大流数量骤减时减少分段, 以提高存储空间利用率. 综合上述思路, 本文提出了一种支持流量抖动的网络大流弹性识别方法 RobustSketch, 如图 1 所示.

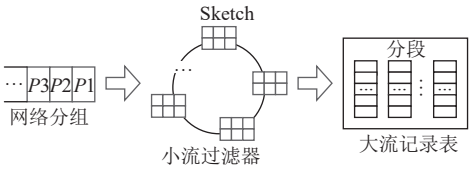


图 1 网络大流弹性识别方法 RobustSketch

RobustSketch 由小流过滤器和大流记录表两部分组成. 小流过滤器由数量动态调整的若干个相同 Sketch 循环链接而成. 每个 Sketch 是由只占少数比特的计数器构成的二维向量, 用于记录当前周期传入的分组数量. Sketch 数量可根据分组到达速率的动态变化进行适应性扩展与收缩, 以确保记录给定时间周期内的所有分组, 进而始终精确过滤小流. 每当 Sketch 循环链进行伸缩操作时, 会根据循环链中的所有 Sketch 加权平均生成一个统计 Sketch. 对于一条传入流, 仅当其映射到统计 Sketch 中的所有计数器达到阈值时, 才允许进入大流记录表, 以进一步精确统计流的分组数量, 从而达到精确跟踪大流的目的. 大流记录表采用动态可伸缩的多分段哈希表, 每个哈希桶包含标识流的流指纹和记录分组数量的计数器. 对于每条传入流, 大流记录表提供多个候选位置, 以降低哈希冲突率, 并结合概率衰减策略剔除潜在小流, 以提高大流识别准确率. 此外, 大流记录表可根据网络大流数量的动态变化增加或减少分段, 以尽可能记录所有大流, 同时提高存储空间利用率.

在上述方法中, 对于一个传入的分组, 首先将其插入小流过滤器中当前计数的 Sketch 进行记录. 然后根据其流标识符 *fid* 在统计 Sketch 每行中映射一个位置, 进而读取对应的计数器值. 若映射位置中最小计数器值未达到预设阈值, 则判定分组属于小流, 并直接丢弃, 否则将分组所属的流插入大流记录表, 以进一步筛选. 在大流记录表中, 分组在每个分段中通过对应的子哈希算法映射一个位置, 进而并行查找其中是否存有对应的流. 若查找成功, 则将该流的分组数量加 1; 若查找失败, 且候选位置中存在空位, 则将该流存入一个空位, 同时将其计数器置 1. 否则, 找出映射位置中的最小流, 进而根据其记录的分组数量对其计数器进行概率衰减. 若衰减到 0, 则将传入分组所属的新流替换最小流, 否则丢弃传入分组.

3.2 小流过滤器

3.2.1 小流过滤器结构

传统的过滤器使用二维计数器数组记录流到达的分组数量, 且当流映射到的所有计数器的值达到过滤阈值 α 时, 允许其通过. 然而, 随着网络分组的持续到达, 尤其是当网络流量激增时, 过多的小流产生的大量哈希冲突将导致大部分计数器快速达到阈值, 进而造成小流过滤器逐步趋于失效. 对此, 本文设计了一种循环结构的可伸缩 Sketch 过滤器, 如图 2 所示. 该过滤器可根据分组到达速率的动态变化适应性增加与减少 Sketch 数量, 以确保记录给定时间周期内的所有分组, 从而解决过滤器溢出失效的问题. 进一步, 基于 Sketch 循环链, 采用加权平均判断策略, 设计统计 Sketch 作为小流过滤的依据, 有效保证小流过滤的准确性.

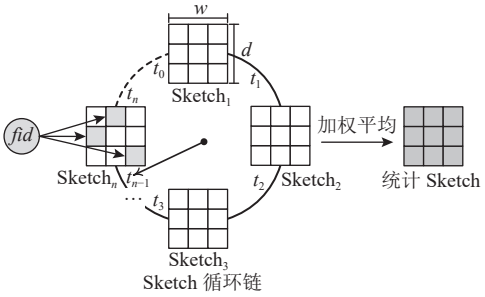


图 2 小流过滤器数据结构

在 Sketch 循环链中, 每个 Sketch 由 d 行 w 列个计数器组成, 每行计数器与一个独立的哈希函数 $h_i(\cdot)$ 关联, 每个计数器记录映射到该位置的分组数量. 每个 Sketch 记录开始时间和结束时间, 作为当前计数的 Sketch 饱和时判断循环链扩展或收缩的依据. 每个到达的分组通过 d 个哈希函数 $h_1(\cdot)$, $h_2(\cdot)$,…, $h_d(\cdot)$, 在当前计数 Sketch 的每行中映射一个计数器, 以完成计数过程. 随着网络分组的持续到达, 当前计数 Sketch 中达到阈值的计数器不断增多, 逐步趋于饱和. 当达到饱和状态时, 根据循环链中的所有 Sketch 生成新的统计 Sketch, 作为判断传入分组是否属于小流的依据. 同时, 根据循环链中每个 Sketch 的起止时间对循环链进行扩展或收缩, 使 Sketch 循环链完整记录固定时间周期内的所有分组, 从而在分组到达速率波动的情况下, 仍能取得准确过滤小流的效果.

3.2.2 小流过滤流程

当小流过滤器收到一个分组的流标识符 *fid* 时, 首先用 d 个不同的哈希函数在当前计数 Sketch 的每个桶中映射一个计数器, 然后在所有映射计数器中找出最小计数器 $FC_{\min}$. 若 $FC_{\min}$ 未达到过滤阈值 α, 则将其使其加 1, 从而完成对该分组的计数. 若当前 Sketch 中达到阈值的计数器数量 N_0 超过预设更新比例 σ, 则更新统计 Sketch, 并执行 Sketch 循环链伸缩流程. 接下来, 在统计 Sketch 的每个桶中映射一个计数器, 并找出其中最小的计数器 $SC_{\min}$. 如果最小计数器未达到过滤阈值 α, 则确定为小流, 否则判定为候选大流并插入大流记录表.

算法 1. 小流过滤器插入算法 (Flow ID).

输入: 待插入流 f, 过滤器阈值 α, 预设更新比例 σ;
输出: 是否成功插入.

1. $FC_{\min} \leftarrow \min_{1\leqslant i\leqslant d}(F[i][h_i(f)])$; //$F$ 表示当前计数 Sketch
2. **if** $FC_{\min} < \alpha$ **then**
3. **for** $i \leftarrow 1, d$ **do**
4. **if** $F[i][h_i(f)] == FC_{\min}$ **then**
5. $F[i][h_i(f)] \leftarrow F[i][h_i(f)]+1$;
6. **if** $F[i][h_i(f)] == \alpha$ **then**
7. $N_0 \leftarrow N_0+1$;
8. **if** $N_0/w \times d \geqslant \sigma$ **then**
9. UpdateStatisticSketch(); //统计 Sketch 更新
10. StretchFilter(); //Sketch 循环链伸缩
11. $SC_{\min} \leftarrow \min_{1\leqslant i\leqslant d}(S[i][h_i(f)])$; //$S$ 表示当前统计 Sketch
12. **if** $SC_{\min} \geqslant \alpha$ **then**
13. InsertRecorder(f); //候选大流插入
14. **return** true;

举例: 图 3 给出了小流过滤器的插入示例. 假设小流过滤器的 $w=d=3$, 总计数器数量 $N_c=w\times d=9$, 过滤阈值 $\alpha=15$, 过滤器更新的预设比例 $\sigma=70\%$. 给定每个传入的分组, 我们计算 3 个不同的哈希函数, 以在当前计数 Sketch 的每行中映射一个计数器. 1) 对于流 id 为 f_1 的传入分组, 其在当前计数 Sketch 映射计数器中的最小计数器值为 $FC_{\min}=\{3, 15, 3\}=3$. 由于 $3<\alpha=15$, 因此将 $FC_{\min}$ 的值加 1, 即 $FC_{\min}+1=3+1=4$. 然后, 我们使用同样的 3 个不同哈希函数将 f_1 映射到统计 Sketch 中的 3 个计数器, 找出最小计数器值为 $SC_{\min}=\{10, 6, 5\}=5$. 因为 $5<\alpha=15$, 所以判定流 f_1 为小流, 不予放行. 2) 对于流 id 为 f_2 的传入分组, 其在当前计数 Sketch 映射计数器中的最小计数器值为 $FC_{\min}=\{14, 15, 15\}=14$. 由于 $14<\alpha=15$, 因此将 $FC_{\min}$ 的值加 1, 即 $FC_{\min}+1=14+1=15=\alpha$. 此时, 需要增加达到阈值的计数器数量, 即 $N_0+1=4+1=5$. 由于 $5<N_c\times\sigma$, 表明当前计数 Sketch 尚未饱和, 无需进行统计 Sketch 更新. 接着, 我们将 f_2 映射到统计 Sketch 中, 映射计数器中的最小计数器值为 $SC_{\min}=\{15, 15, 15\}=15$. 因为 $15=\alpha$, 表明流 f_2 可能是一条大流, 因此将流 f_2 放行至大流记录表.

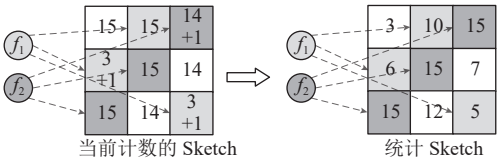


图 3 小流过滤器插入示例

3.2.3 统计 Sketch 更新

若当前计数 Sketch 已饱和, 即达到阈值的计数器个数超过一定比例, 则进入统计 Sketch 更新流程. 首先获取循环链中 Sketch 的行数 d, 列数 w, 以及数量 N_f. 然后将循环链中的第 k 个 Sketch 赋予权重 $2^{k-1}/(2^n-1)$, 进而将每个 Sketch 中同位置的计数器进行加权平均, 得到统计 Sketch 中对应计数器的新值.

3.2.4 Sketch 循环链伸缩

在统计 Sketch 更新结束后将对 Sketch 循环链进行伸缩. 首先获取循环链中的 Sketch 数量 N_f, 第 1 个 Sketch 的开始时间 t_0, 以及当前时刻 t. 若 t 距离 t_0 的时间长度小于给定的时间周期 T, 则新增一个 Sketch, 作为新计数 Sketch 加入循环链尾部. 否则依次获取第 i ($1\leqslant i\leqslant n-1$) 个 Sketch 的结束时刻 t_i, 并判断 t 距离 t_i 的时间长度是否小于 T. 若不是, 则删除该 Sketch. 依次循环, 直到 t 距离 t_i 的时间长度小于 T 为止. 最后, 重置第 i 个 Sketch 继续计数.

举例: 图 4 给出了 Sketch 循环链伸缩示例. 假设当前时间为 t, 时间周期为 T, 由于当前计数的 Sketch 饱和时立刻执行伸缩操作, 所以上一个 Sketch 的结束时间等于下一个 Sketch 的开始时间. 1) Sketch 循环链伸展示例如图 4(a) 所示, 此时循环链中有 3 个 Sketch 且当前计数的 Sketch 为 $Sketch_3$. 在时间点 t 时, $Sketch_3$ 收到一个传入包在处理后达到伸缩条件. 计算得到当前时间 t 与 $Sketch_1$ 的开始时间 t_0 之间的间隔 $t-t_0<T$, 则直接 Sketch 循环链的末尾新增一个 $Sketch_4$, 并将 $Sketch_3$ 的结束时间 $S_3.t_{end}$ 与 $Sketch_4$ 的开始时间 $S_4.t_{start}$ 赋值为当前时间 t, $Sketch_4$ 的结束时间 $S_4.t_{end}$ 应该等到其达到伸缩条件时再设定. 2) Sketch 循环链收缩示例如图 4(b) 所示, 此时循环链中有 4 个 Sketch 且当前计数的 Sketch 为 $Sketch_4$. 在时间点 t 时, $Sketch_4$ 收到一个传入包在处理后达到伸缩条件. 计算得到当前时间 t 与 $Sketch_1$ 的开始时间 t_0 之间的间隔 $t-t_0>T$, 不满足伸展条件. 接着获取 $Sketch_1$ 的结束时间 t_1, 计算得到当前时间 t 与 t_1 之间的间隔 $t-t_1>T$, 满足收缩条件, 于是将 $Sketch_1$ 从循环链中删除. 然后, 获取第 $Sketch_2$ 的结束时间 t_2, 计算得到当前时间 t 与 t_2 之间的间隔 $t-t_2<T$, 所以重置 $Sketch_2$ 的所有计数器值, 将 $Sketch_2$ 的开始时间 $S_2.t_{start}$ 设置为 t, 并重置结束时间, 然后链接到循环链尾部.

3.3 大流记录表

3.3.1 大流记录表结构

传统的大流记录表需要在测量前设定容量且测量过程中无法改变. 在流量发生抖动时, 网络大流数量急剧增加, 很容易出现哈希桶满载的情况, 导致后续到达的大流无法存入而被丢弃, 使得大流识别的准确性严重下降. 对此, 本文设计一种基于动态分段哈希的可拓展大流记录表, 可根据记录表中的大流占比自适应增加或减少分段数

量, 以始终完整记录所有大流, 从而保证大流识别的高准确率. 当一条新的候选大流进入大流记录表时, 若对应的候选位置均已满, 需要决定是否替换候选位置中的某条流. 对此, 本文设计基于概率衰减的候选大流替换策略, 以一定概率衰减候选位置中的最小流, 并在衰减为零时替换为新候选大流, 以尽可能保存大流, 从而提高大流识别精度. 大流记录表的数据结构如图 5 所示.

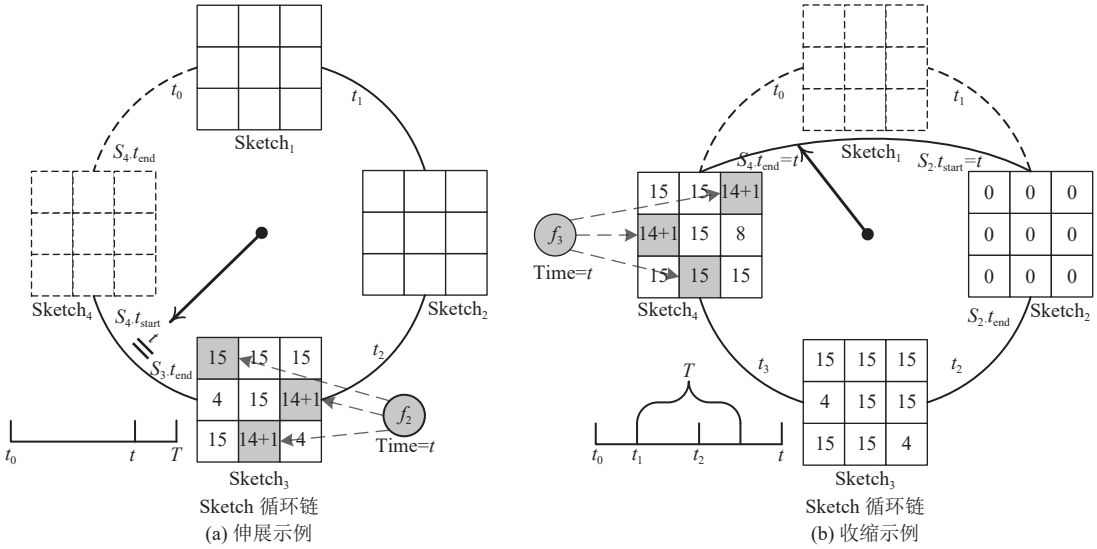


图 4 Sketch 循环链伸缩示例

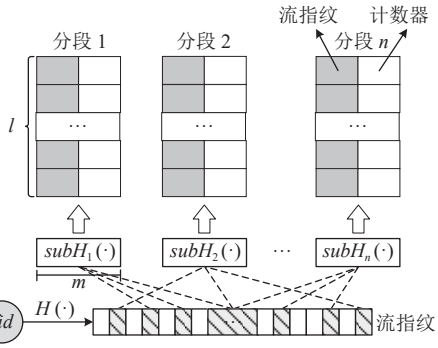


图 5 大流记录表的数据结构

大流记录表是由数量动态可变的多个逻辑分段组成的可伸缩哈希表. 每个分段由固定数量的大流项组成, 并通过对应的子哈希函数 $subH_i(\cdot)$ 索引. 每个大流项包含对应的流指纹 *fp* 和记录分组数量的计数器, 其中流指纹 *fp* 由流标识符 *fid* 通过哈希函数生成. 假设分段长度为 l, 子哈希函数首先从流指纹 *fp* 随机选取不同的 m ($m \geqslant \log_2 l$) 个比特位, 并随机排列成 m 位比特串, 然后对 l 取模. 当大流记录表中达到分组数量阈值的大流占比过多或过少时, 则新增或减少分段, 以适应网络流量的抖动性. 当一条新的候选大流进入大流记录表时, 若所有索引位置均已满, 则找出其中分组数量最少的大流项, 然后以一定概率对其计数器进行衰减. 若计数器衰减到 0, 则将新候选大流替换最小流, 以尽可能将大流保留下来, 并及时剔除小流.

3.3.2 候选大流插入操作

当一个属于流 f 的分组进入大流记录表后, 首先根据其流标识符 *fid* 生成流指纹 *fp*, 然后在每个分段中通过对应的子哈希函数映射一个大流项, 进而通过流指纹进行匹配. 匹配结果及处理分为 3 种情况.

(1) 若 *fp* 成功匹配一个映射的大流项, 则将匹配大流项的计数器加 1. 若增加后计数器值达到预设大流阈值 β,

则将总大流数 M_0 加 1. 此时, 若大流记录表中的总大流占比过高, 则新增一个空分段, 以记录后续到达的大流.

(2) 若 *fp* 与所有映射的大流项均不匹配, 则检查映射大流项中是否存在空项. 若存在, 则将新流插入其中, 并将计数器置为 1.

(3) 若 *fp* 与所有映射的大流项均不匹配, 且所有映射大流项均不为空, 则找出所有映射大流项中的最小计数器, 并以概率 b^{-C} 使其减 1. 其中, C 为计数器值, b 为略大于 1 的常数. 若计数器衰减后其值低于预设大流阈值 β, 则将总大流数 M_0 减 1. 此时, 若大流记录表中的总大流占比过低, 则删除一个分段, 以降低存储空间开销. 当计数器减为 0 时, 将新流插入其中, 并将计数器置为 1.

算法 2. 大流记录器插入算法.

输入: 待插入流 *f*, 过滤器阈值 α, 大流阈值 β, 扩增阈值 λ, 收缩阈值 η, 衰减指数 b;
输出: 是否插入成功.

```
1. fp ← Genfingerprint(f); //计算流指纹
2. i_min ← 1, pos_min ←SubH_1(fp);
3. C_min ←R[i_min][ pos_min].count;
4. for i ← 1, n do //分段数为 n
5.     pos ← SubH_i(fp);
6.     if R[i][pos].fp==fp then
7.         R[i][pos].count ← R[i][pos].count+1;
8.         if (R[i][pos].count+α)==β then
9.             M_0 ←M_0+1;
10.            if (M_0/nl)>λ then
11.                StretchRecoder(true); //记录表扩展
12.        return true;
13.    if R[i][pos].fp==null then
14.        R[i][pos].fp ← fp;
15.        R[i][pos].count ← 1;
16.        return true;
17.    if R[i][pos].count < C_min
18.        i_min ← i, pos_min ← pos, C_min ← R[i][pos].count;
19. x ← rand()/RAND_MAX; //生成随机数
20. y ← pow(b, –C_min);
21. if x<y then
22.    R[i_min][pos_min].count ← R[i_min][pos_min ].count–1;
23.    if (R[i_min][pos_min].count+α)==β–1 then
24.        M_0 ←M_0–1;
25.        if (M_0 / nl)<η then
26.            StretchRecoder(false); //记录表收缩
27.    if Recorder[i_min][pos_min].count==0 then
28.        Recorder[i_min][ pos_min].fp ←fp;
29.        Recorder[i_min][ pos_min].count ← 1;
30. return true;
```

举例: 图 6 给出了大流记录表的插入示例. 假设大流记录表有 3 个分段, 每个分段长度为 13, 指纹长度 m 为 8, 子哈希函数选取的比特位数 n 为 4, 概率衰减底数为 1.08. 给定每个传入的分组, 我们计算 3 个子哈希函数在每个分段映射一个大流项, 然后并行的查找所有大流项. 1) 插入流指纹为 fp_5 的流 f_5. 所有映射的大流项中的指纹值均与 fp_5 不匹配, 但存在空大流项. 因此选择第 1 个空大流项将 fp_5 插入其中, 并将计数器值置为 1. 2) 插入流指纹为 fp_6 的流 f_6. 存在一个大流项中的指纹值与 fp_6 相匹配, 说明 f_6 已记录在大流记录表中. 因此, 我们将其对应的计数器加 1 (5+1=6). 3) 插入流指纹为 fp_8 的流 f_8. 所有映射的大流项中的指纹值均与 fp_8 不匹配, 且不存在空大流项. 此时, 我们在所有映射大流项中找出计数器值最小的大流项<fp_9, 1>, 并以 1.08^{-1} 的概率将其计数器值减 1. 假设衰减成功, 则计数器减为 0, 此时将<fp_8, 1>替换入该大流项.

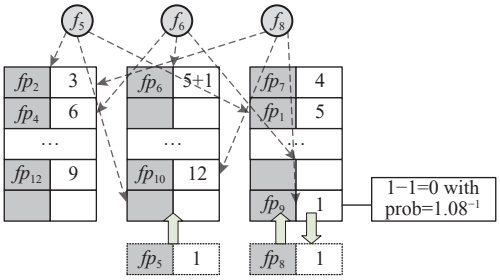


图 6　大流记录表插入示例

3.3.3　记录表伸缩

当大流记录表中的大流占比超出预设上限, 则新增一个空分段, 并生成新的子哈希函数算法用于索引该分段. 当大流记录表中的大流占比低于预设下限, 则首先获取大流记录表中的分段数量 n, 并找出所有分段中存储大流数量最少的分段, 作为待删除分段. 然后, 从该分段存储的第一条流开始, 依次将该分段中的每条流按其流指纹 fp 在其余分段中各映射一个大流项. 若存在空大流项, 则将该流插入其中. 若不存在空大流项, 则找出所有映射大流项中的最小计数器值 $RC_{\min}$, 并与该流记录的分组数量进行比较. 若较小, 则将该流换入最小计数器对应的大流项中, 否则直接丢弃. 待该分段中的所有流均处理完毕后, 删除该分段.

举例: 图 7 给出了大流记录表伸缩示例. 假设每个分段长度为 5, 大流阈值为 50, 分段扩增阈值为 30%, 分段收缩阈值为 20%. 1) 大流记录表分段扩增示例如图 7(a) 所示, 此时大流记录表中的分段数为 3, 一个流 id 为 f_5 的传入分组, 通过 3 个子哈希函数在每个分段映射一个大流项, 并行查找所有大流项, 映射大流项中存在与流 f_5 相同的指纹值, 因此将其对应的计数器加 1 (49+1=50≥β). 此时大流记录表中存储的大流数量变为 5 条>3×5×30%, 满足扩增条件, 于是新增一个空分段, 并根据第 3.3.1 节所述方式生成新的子哈希函数用于索引该分段. 2) 大流记录表分段收缩示例如图 7(b) 所示, 此时大流记录表中的分段数为 4, 一个流 id 为 f_{16} 的传入分组, 通过 4 个子哈希函数在每个分段映射一个大流项, 并行查找所有大流项, 映射大流项中不存在与流 f_{16} 相同的指纹值, 且不存在空大流项. 此时找出映射大流项中的最小计数器<fp_{14}, 50>, 并以 1.08^{-50} 的概率将其减 1. 若衰减成功, 此时大流记录表中的大流数量由 4 减为 3 条<4×5×20%, 满足收缩条件, 此时选择分段 4 作为待删除分段, 并将分段 4 中存储的流 f_{15}、f_{14} 重新插入前 3 个分段中. 其中流 f_{15} 在映射大流项中找到空项直接插入, 流 f_{14} 在前 3 个分段中的映射大流项均已满, 但其记录的分组数量 49 大于映射大流项中的最小计数器<fp_1, 5>, 于是用<fp_{14}, 49>替换入该大流项位置.

3.3.4　大流报告

在进行大流识别时, 首先获取大流记录表的分段数量 n 和分段长度 l. 然后在每个分段中逐个检查大流项中的计数器. 若计数器达到阈值 β, 则输出对应的流指纹和流大小, 即计数器值与小流过滤阈值之和.

3.4　理论分析

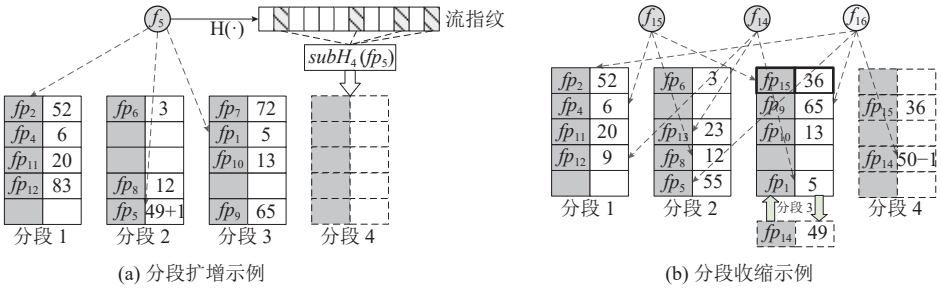
3.4.1　小流过滤器的误差界限

定理 1. 给定包含 L 条流的分组序列 $S=(p_1, p_2, \ldots, p_N)$, 令 f_i 表示第 i 条流真实出现的频率, $\hat{f}_i$ 表示第 i 条流在

小流过滤器中的估计值, 假设小流过滤器中的每个 Sketch 有 d 行 w 列, 则小流过滤器的假阳性错误率 FPR 满足:

$$FPR \leqslant \frac{\varepsilon N^2}{e\ln(1/\delta)} + \delta \tag{1}$$

其中, $\delta,\varepsilon \in (0,1)$, 且满足关系 $d = \left\lceil \ln\left(\frac{1}{\delta}\right) \right\rceil$, $w = \left\lceil \frac{e}{\varepsilon} \right\rceil$.



(a) 分段扩增示例 (b) 分段收缩示例

图 7 大流记录表伸缩示例

证明: 由前文所述可知, 小流过滤器和统计 Sketch 以 CM Sketch 为基础, 而 CM Sketch 存在如下误差界限[28]:

$$P\left(\hat{f}_i - f_i > \sigma N\right) \leqslant \delta \tag{2}$$

接下来证明小流过滤器的误差界限. 假设小流过滤器阈值为 α, 若第 i 条流在统计 Sketch 中的估计频率 $\hat{f}_i > \alpha$, 则认为该流可能是一条大流, 并插入大流记录表. 因此, 小流过滤器的误差界限可表示为:

$$FPR = P\left(\hat{f}_i > \alpha \cap f_i < \alpha\right) \tag{3}$$

令公式 (2) 中的 $\sigma = 1/2$, 根据 Chebyshev 不等式, 有:

$$P\left(\hat{f}_i - f_i > \frac{N}{2}\right) \leqslant \frac{1}{4\varepsilon^2 N^2} \tag{4}$$

根据概率的加减法和反转不等式, 可得:

$$FPR = P\left(\hat{f}_i > \alpha \cap f_i < \alpha\right) \leqslant P\left(\hat{f}_i - f_i > \frac{N}{2}\right) + P\left(f_i < \hat{f}_i - \alpha\right) \tag{5}$$

将公式 (4) 代入公式 (5) 得到:

$$P\left(\hat{f}_i - f_i > \frac{N}{2}\right) + P\left(f_i < \hat{f}_i - \alpha\right) \leqslant \frac{1}{4\varepsilon^2 N^2} + \frac{\sum_{i=1}^{M} P(f_i < \alpha)}{wd} \leqslant \frac{N^2}{wd} + \delta \tag{6}$$

因为 $d = \left\lceil \ln\left(\frac{1}{\delta}\right) \right\rceil$, $w = \left\lceil \frac{e}{\varepsilon} \right\rceil$, 所以有:

$$\frac{N^2}{wd} + \delta = \frac{N^2}{\lceil e/\varepsilon \rceil \cdot \lceil \ln(1/\delta) \rceil} + \delta \tag{7}$$

又因为 $x \leqslant \lceil x \rceil \leqslant x + 1$, 所以有:

$$\frac{N^2}{\lceil e/\varepsilon \rceil \cdot \lceil \ln(1/\delta) \rceil} \leqslant \frac{N^2}{(e/\varepsilon) \cdot \ln(1/\delta)} \tag{8}$$

根据公式 (6)–公式 (8), 最终得到:

$$FPR \leqslant \frac{N^2}{(e/\varepsilon) \cdot \ln(1/\delta)} + \delta = \frac{\varepsilon N^2}{e\ln(1/\delta)} + \delta \tag{9}$$

3.4.2 大流记录表的误差界限

定理 2. 假定网络中有 M 条大流, f_i 表示第 i 条大流的真实频率, $\hat{f}_i$ 表示第 i 条大流在大流记录表中的估计频率. 设大流记录表的分段数为 n, 分段长度为 l. 则大流记录表的大流频率估计误差满足:

$$P\left(\left|\hat{f}_i - f_i\right| > \frac{bf_i}{\sqrt[n]{l}}\right) \leqslant \frac{1}{l^b} \tag{10}$$

其中, b 为概率衰减底数.

证明: 首先, 根据大流记录表的插入过程可知, 在插入一个属于流 i 的数据分组时, 将通过哈希函数映射 d 个哈希桶, 并检查这些桶中是否已存储该流. 如果所有的映射桶中都没有存储传入流, 则选择 d 个哈希桶中记录的最小流频率 $\hat{f}_i$, 以 $b^{-\hat{f}_i}$ 的概率将其减 1. 由此可得, 任意一条流 i 的计数器值在任何时候都满足以下不等式:

$$\hat{f}_i \geqslant \log_b^{f_i} \tag{11}$$

假设分组插入时被映射到的桶集合为 $S=\{s_1, s_2,\ldots, s_d\}$, 则该流的计数器更新公式可表示为:

$$\hat{f}_i \leftarrow \hat{f}_i + 1 - \min_{s\in S} b^{-\hat{f}_s} \tag{12}$$

接下来证明误差界限. 对于一条真实频率为 f_i, 大流记录表中计数器值为 $\hat{f}_i$ 的大流 i, 令 $k\geqslant 0$, $t=kf_i/l$ 表示插入前 k 个数据分组时存储流 i 的哈希桶被映射到的次数, X_t 表示第 k 个数据分组插入时存储流 i 的哈希桶更新的概率, 则有:

$$X_t = \Pr\left[\hat{f}_i + 1 - \min_{s\in S} b^{-\hat{f}_s} \geqslant t+1\right] - \Pr\left[\hat{f}_i - \min_{s\in S} b^{-\hat{f}_s} \geqslant t\right] \tag{13}$$

根据大流记录表的插入过程, 在插入一个属于流 i 的数据分组时:

(1) 若 n 个映射大流项中已存储该流, 则将对应的计数器加 1, 不需要进行衰减, 此时 $X_t = 1$.

(2) 若 n 个映射大流项中未存储该流, 但存在空大流项. 则该分组将插入到第 1 个空大流项, 并将其计数器置为 1, 也不会发生概率衰减, 此时 $X_t = (1-b^{-t})$.

(3) 若 n 个映射大流项中未存储该流, 且不存在空大流项, 此时将执行概率衰减策略. 令 $A=\{j\in[1,M], j\neq i\}$ 表示除 i 以外的大流集合. 由于 b>1, 则有:

$$\min_{s\in S} b^{-\hat{f}_s} \geqslant b^{-\max_{j\in A}\hat{f}_j} \tag{14}$$

将公式 (14) 代入公式 (13) 可得:

$$X_t = \Pr\left[\hat{f}_i + 1 - b^{-\max_{j\in A}\hat{f}_j} \geqslant t+1\right] - \Pr\left[C_i - b^{-\max_{j\in A}\hat{f}_j} \geqslant t\right] \tag{15}$$

根据 Chernoff 界限可得:

$$\Pr\left[|X_t - E[X_t]| \geqslant \lambda\right] \leqslant 2\mathrm{e}^{-2\lambda^2} \tag{16}$$

其中, $E[X_t]$ 可以用概率衰减策略计算:

$$\begin{aligned} E[X_t] &= \Pr\left[\hat{f}_i + 1 - b^{-\max_{j\in A}\hat{f}_j} \geqslant t+1\right] - \Pr\left[\hat{f}_i - b^{-\max_{j\in A}\hat{f}_j} \geqslant t\right] = \Pr\left[b^{\hat{f}_i} \geqslant b^{t+1} + b^{\max_{j\in A}\hat{f}_j - 1}\right] - \Pr\left[b^{C_j} \geqslant b^t + b^{\max_{j\in A}\hat{f}_j - 1}\right] \\ &= b^{-t-1}\left(1 - b^{\max_{j\in A}\hat{f}_j - t - 1}\right) - b^{-t}\left(1 - b^{\max_{j\in A}\hat{f}_j - t}\right) \end{aligned} \tag{17}$$

由于总共插入的数据分组为 N, 因此 $k=\dfrac{N}{l}$. 由 Markov 不等式可得:

$$\Pr\left[\left|\hat{f}_i - \frac{kf_i}{l}\right| \geqslant \lambda\right] \leqslant \frac{E\left[\left|\hat{f}_i - \dfrac{kf_i}{l}\right|\right]}{\lambda} \tag{18}$$

考虑到 f_i 是大流的真实频率, 且有 $\sum_{i=1}^{M} f_i = N$, 因此:

$$E\left[\left|\hat{f}_i - \frac{kf_i}{l}\right|\right] = \sum_{t=0}^{\infty} |X_t| \cdot \Pr\left[t \leqslant \hat{f}_i - \frac{kf_i}{l} < t+1\right] \tag{19}$$

将公式 (15) 求出的 X_t 代入公式 (19), 注意到 $\Pr\left[t \leqslant \hat{f}_i - \dfrac{kf_i}{l} < t+1\right]$ 是一个几何分布, 有:

$$E\left[\left|\hat{f}_i - \frac{kf_i}{l}\right|\right] = \sum_{t=0}^{\infty} |X_t| \cdot \frac{f_i}{l}\left(1-\frac{f_i}{l}\right)^t \leqslant \sum_{t=0}^{\infty} |X_t| \cdot \frac{f_i}{l}\left(\frac{1}{b}\right)^{\left|\hat{f}_i - \frac{kf_i}{T}\right|} \tag{20}$$

由于 X_t 是非负的, 令 $\lambda = \dfrac{bf_i}{\sqrt[d]{l}}$, 代入公式 (20) 可得:

$$E\left[\left|\hat{f}_i - \frac{kf_i}{l}\right|\right] \leqslant \frac{1}{\lambda}\sum_{t=0}^{\infty} X_t \cdot \frac{f_i}{l}\left(b^{1/d}\right)^{-t} \tag{21}$$

将公式 (15) 代入公式 (21), 并使用几何级数公式可得:

$$E\left[\left|\hat{f}_i-\frac{kf_i}{l}\right|\right]\leqslant\frac{1}{\lambda}\left(1+\frac{1-b^{-f_i/l_i}}{b^{1/d}-1}\right)\cdot\frac{bf_i}{\sqrt[d]{l}} \tag{22}$$

代入 Chernoff 界限可得:

$$Pr\left[\left|\hat{f}_i-\frac{kf_i}{l}\right|\geqslant\frac{bf_i}{\sqrt[d]{l}}\right]\leqslant\frac{(b^{1/d}+1)(1-b^{-f_i/w_i})}{b}\cdot\frac{1}{l^b} \tag{23}$$

将 $f_i>1$ 和 $b>1$ 代入公式 (23) 可得:

$$P\left(\left|\hat{f}_i-f_i\right|>\frac{bf_i}{\sqrt[n]{l}}\right)\leqslant\frac{1}{l^b} \tag{24}$$

4 实 验

4.1 实验方法

(1) 实验平台配置. 双 6 核 CPU (24 个线程, Intel Xeon Silver 4214R @ 2.4 GHz) , 32 GB DRAM 内存, 每个 CPU 有三级缓存: 两个 32 KB 的 L1 数据缓存 (1 个指令缓存和 1 个数据缓存), 256 KB 的 L2 数据缓存和 24 MB 的共享 L3 数据缓存.

(2) 数据集. 实验选取江苏省计算机网络技术重点实验室发布的两个高速网络流量样本 (TRACE20130222 和 TRACE20181206) 作为测试数据集. 这两个样本采集于中国教育科研网江苏省边界 10 Gb/s 主干链路, 采样比例为 1:4, 采集日期分别为 2013 年 02 月 22 日和 2018 年 12 月 06 日. 样本 TRACE20130222 共计 1 204 372 条流, 其中分组数量超过 500 的大流有 2 631 条. 样本 TRACE20181206 共计 686 836 条流, 其中分组数量超过 500 的大流的有 3 147 条.

(3) 实验指标.

1) 精确率 (precision rate, *PR*): 测量算法报告的所有大流中真实大流的占比.

2) *F*1 分数: $\frac{2\times PR\times RR}{PR+RR}$. 其中 *PR* 表示大流数量识别的精确率, *RR* 表示大流数量识别的召回率, 即所有真实大流中被测量算法正确报告的比例.

3) 平均绝对误差 (average absolute error, *AAE*): $\frac{1}{|\varphi|}\sum_{f_i\in\varphi}|\hat{n}_i-n_i|$. 其中 φ 为算法报告的总大流数, $\hat{n}_i$ 为 φ 中第 i 条流的测量大小, n_i 为第 i 条流的真实大小.

4) 平均相对误差 (average relative error, *ARE*): $\frac{1}{|\varphi|}\sum_{f_i\in\varphi}\frac{|\hat{n}_i-n_i|}{n_i}$. 其中 φ 为算法报告的总大流数, $\hat{n}_i$ 为 φ 中第 i 条流的测量大小, n_i 为第 i 条流的真实大小.

(4) 参数设置. 本文采用 C++编程实现现有的经典测量方法 Elastic Sketch、Augmented Sketch (ASketch)、Cold Filter+CM Sketch (CF+CM), 以及本文所提出的 RobustSketch. 对于 Elastic Sketch 方法, 轻部分由一个计数器数组组成, 重部分的每个桶存储 7 条流和 1 个反票计数器, 替换阈值设为 8, 存储桶的数量取决于剩余内存大小. 对于 ASketch 方法, 按比例 1:3 分配 Filter 部分和 CM Sketch 部分的大小, 其中 CM Sketch 的行数设为 3, CM Sketch 的列数和 Filter 部分的桶数量取决分配的内存大小. 对于 CF+CM 方法, 第 1 层过滤阈值设为 15, 第 2 层过滤阈值设为 255, CM Sketch 行数设为 8, 列数取决于剩余内存大小.

对于本文提出的 RobustSketch, 过滤器中的每个 Sketch 设为 3 行 1 024 列, 过滤阈值 α 设为 15, 小流过滤器更新阈值 σ 设为 85%, 时间周期设为 500 μs; 大流记录表初始分段数设为 8, 最大分段数取决于剩余内存大小, 伸展阈值为 25%, 收缩阈值为 15%, 概率衰减底数 b 设为 1.08. 接下来, 以数据集 TRACE20181206 为例, 部分核心参数设置原因如下:

1) 过滤阈值. 在保持其余参数不变的情况下, 依次将过滤阈值从 2^3-1 增加到 2^6-1, 统计 RobustSketch 各项性能指标如表 1 所示. 从表 1 可以看出: 当过滤阈值为 7 时, RobustSketch 取得了不错的精确率但平均绝对误差最高.

这是因为当过滤阈值较低时, 小流过滤器会放行大量流进入大流记录表, 其中可能包含许多小流, 造成哈希冲突从而导致对大流的估计值出现误差. 随着过滤阈值的增大, RobustSketch 的精度会有稍许下降. 这是因为大流在单个时间周期内的分组数量有限, 若过滤阈设置过高, 小流过滤器可能会过滤掉部分潜在的大流. 当过滤阈值为 15 时, RobustSketch 各项性能指标较好, 且小流过滤器的计数器占用空间少. 综上, 将小流过滤器的计数器大小适合设置为 4 bit, 即过滤阈值为 $2^4-1=15$.

2) 时间周期. 在保持其余参数不变的情况下, 依次将时间周期从 100 μs 增加到 900 μs, 统计 RobustSketch 各项性能指标如表 2 所示. 从表 2 可以看出: 当小流过滤器的时间周期为 500 μs 时, RobustSketch 的精确率最高, *ARE* 最小且优势显著, 而 *F*1 分数和 *AAE* 接近最优值. 此外, 时间周期越短, 循环链中的 Sketch 数量越少, 内存开销越小. 综上, 时间周期适合选取 500 μs.

表 1　小流过滤器过滤阈值对 RobustSketch 性能的影响

过滤阈值	精确率 (%)	*F*1分数	$\log_{10}AAE$	$\log_{10}ARE$
7	98.11	0.918	1.974	**−0.215**
15	**99.85**	**0.924**	1.935	−0.130
31	97.76	0.908	1.914	0.074
63	95.38	0.898	**1.875**	0.354

表 2　小流过滤器时间周期对 RobustSketch 性能的影响

时间周期 (μs)	精确率 (%)	*F*1分数	$\log_{10}AAE$	$\log_{10}ARE$
100	99.83	0.912	1.965	−0.067
300	99.82	**0.926**	1.977	−0.086
500	**99.85**	0.924	1.935	**−0.130**
700	99.75	0.915	1.937	−0.117
900	99.69	0.914	**1.934**	−0.121

3) 大流记录表初始分段数. 在保持其余参数不变的情况下, 依次将大流记录表初始分段数从 2 增加到 10, 统计 RobustSketch 各项性能指标如表 3 所示. 从表 3 可以看出: 在大流记录表初始分段数较低时, 随着初始分段数逐渐增加, RobustSketch 的测量性能整体呈上升趋势. 而当初始分段数由 8 增加到 10 时, 性能变化较小且部分性能指标略微下降. 因此, 将大流记录表初始分段数设为 8.

4) 大流记录表伸缩阈值. 在保持其余参数不变的情况下, 通过调整大流记录表的伸缩阈值, 统计 RobustSketch 各项性能指标如表 4 所示. 从表中可以看出: 当大流记录表伸展阈值为 25%, 收缩阈值为 15% 或伸展阈值为 20%, 收缩阈值为 10% 时, RobustSketch 整体上取得了较好的性能. 在性能相近的情况下, 大流记录表的分段越多, 内存开销越大. 因此, 大流记录表的伸展和收缩阈值分别设置为 25% 和 15%.

表 3　大流记录表初始分段数对 RobustSketch 性能的影响

初始分段数	精确率 (%)	*F*1分数	$\log_{10}AAE$	$\log_{10}ARE$
2	**99.85**	0.856	2.147	−0.074
4	99.81	0.907	2.010	−0.087
6	99.78	0.919	1.958	−0.101
8	**99.85**	**0.924**	1.935	**−0.130**
10	99.74	0.923	**1.891**	−0.124

表 4　大流记录表伸缩阈值对 RobustSketch 性能的影响

伸展 (%)/收缩阈值 (%)	精确率 (%)	*F*1分数	$\log_{10}AAE$	$\log_{10}ARE$
40/30	99.81	0.914	2.001	−0.091
35/25	99.78	0.916	2.005	−0.106
30/20	99.85	0.921	1.974	−0.088
25/15	**99.85**	**0.924**	1.935	**−0.130**
20/10	99.81	0.923	**1.926**	−0.1119

基于上述设置, 从实验数据集中逐个读取数据分组, 分别执行不同的测量算法, 统计算法报告的总大流数和每条大流分组数量, 并结合数据集中的真实大流数和真实大流分组数量, 计算出正确报告的大流数, 进而计算出各种实验指标. 实验分别选取内存大小和分组数量作为参量. 当参量为内存大小时, 测量分组数量固定为 N=10M, 内存大小从 60 KB 递增到 100 KB. 当参量为分组数量时, 内存大小固定 100 KB, 测量分组数量从 6M 递增到 14M.

4.2　大流数量识别精度

4.2.1　精确率

(1) 精确率与内存大小的关系

图 8 展示了不同内存大小下各算法的精确率. 从图 8 可以看出: 在相同的内存使用量下, RobustSketch 相对于 Elastic Sketch、ASketch、CF+CM 具有更高的识别精确率, 且在低内存使用量下仍可取得较高的精确率. 以

TRACE20130222 为例, 当内存使用量为 60 KB 时, Elastic Sketch、ASketch、CF+CM 的精确率分别为 90%、72%、1%, 而 RobustSketch 的精确率达到了 99%. 随着内存使用量的增大, 各算法的识别精确率逐渐提升. 当内存使用量为 100 KB 时, RobustSketch 的精确率接近 100%, 而对于 Elastic Sketch、ASketch、CF+CM, 精确率分别为 98%, 91%, 56%. 总之, RobustSketch 的识别精确率始终最高且能够保持稳定. 这主要是因为 RobustSketch 通过高效的小流过滤器预先筛除了大量小流, 极大地降低了小流分组对大流数量识别的影响, 进而保证了大流数量识别的高准确率.

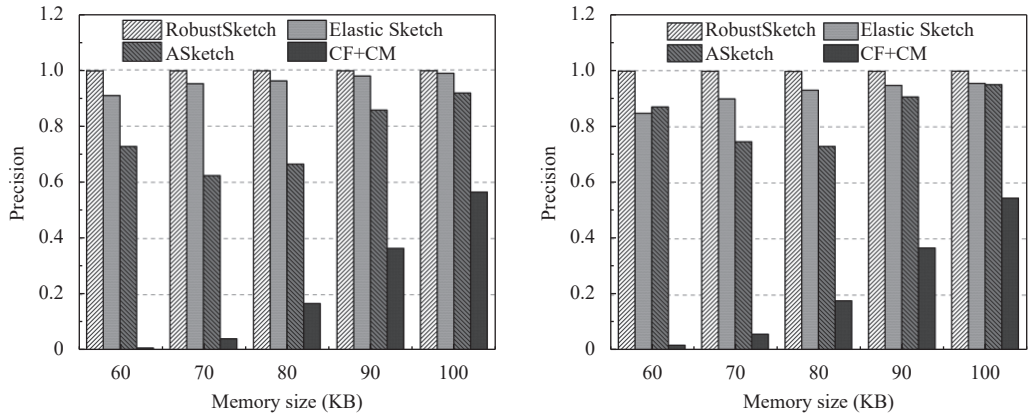


图 8　精确率与内存大小的关系

(2) 精确率与分组数量的关系

图 9 给出了不同分组数量下各算法的精确率变化情况. 从图 9 可以看出: 当内存使用量固定时, 随着测量分组数量增多, RobustSketch 的识别精确率始终保持最高且相对稳定, 而其余算法的精确率呈现不同程度的下降. 以 TRACE20130222 为例, 当测量分组数量为 6M 时, RobustSketch、Elastic Sketch、ASketch、CF+CM 的精确率分别为 99%、99%、97%、90%. 当测量分组数量增加到 14M 时, 各算法的精确率达到最低值, 且 RobustSketch、Elastic Sketch、ASketch、CF+CM 的精确率分别为 99%、96%、54%、20%. 总之, RobustSketch 拥有非常出色的抗流量抖动能力. 这主要归功于 RobustSketch 能够根据网络流量的大小自适应的伸缩大流记录表, 保证对大流的精确存储, 提高了识别精确率.

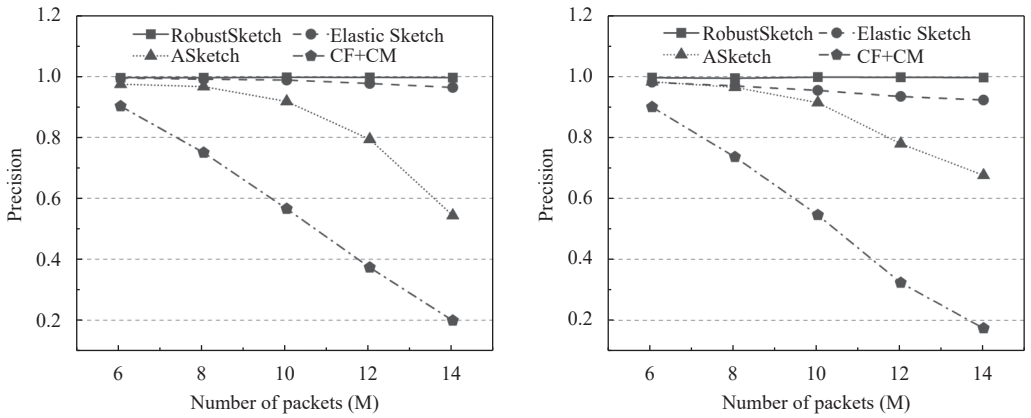


图 9　精确率与分组数量的关系

4.2.2 *F*1 分数

上述精确率由于只考虑了所报告的大流, 而没有考虑被漏报的大流数量, 并不能完全反映大流识别算法的准确性. 除了精确率外, 通常定义召回率为所有真实大流中被测量算法正确报告的比例, 作为大流识别准确性的另一项度量指标. 对于同一个大流识别算法, 召回率往往与精确率呈现负相关关系. 因此, 进一步采用综合精确率和召回率的度量指标 *F*1 分数, 全面评估大流识别的准确性.

(1) *F*1 分数与内存大小的关系

图 10 展示不同内存大小下各算法的 *F*1 分数. 从图 10 可以看出: 在相同的内存使用量下, RobustSketch 的 *F*1 分数比 Elastic Sketch、ASketch、CF+CM 更高, 且在取得相同 *F*1 分数时所使用的内存最小. 以 TRACE20130222 为例, 当内存使用量为 60 KB 时, RobustSketch 的 *F*1 分数达到了 0.92, 而 Elastic Sketch、ASketch、CF+CM 的 *F*1 分数分别为 0.87、0.84、0.02. 当内存使用量增加到 100 KB 时, RobustSketch、Elastic Sketch、ASketch、CF+CM 的 *F*1 分数分别为 0.95, 0.89, 0.93, 0.72. 这主要是因为 RobustSketch 使用低开小的小流过滤器高效过滤小流, 在降低资源开销的同时, 减少了小流与大流之间的哈希冲突, 从而提高了大流识别的准确性.

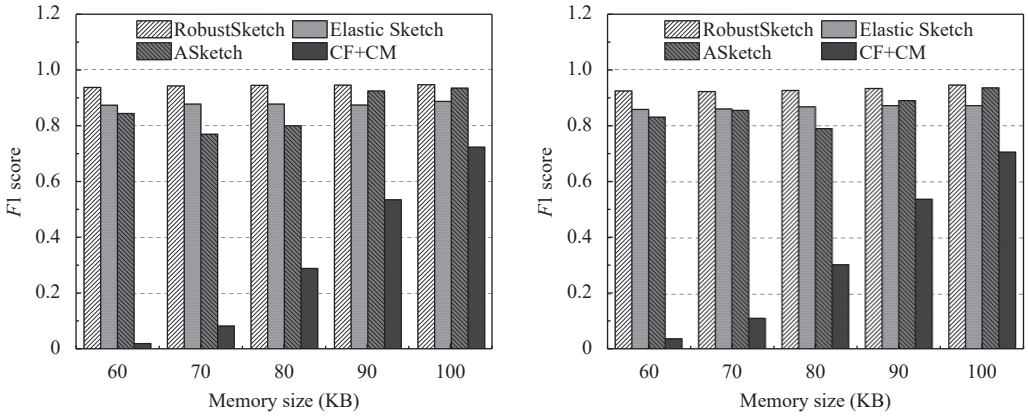


图 10 *F*1 分数与内存大小的关系

(2) *F*1 分数与分组数量的关系

图 11 给出了不同分组数量下各算法的 *F*1 分数变化情况. 从图 11 可以看出: 当内存使用量固定时, 随着测量分组数量增多, RobustSketch 的 *F*1 分数稳定保持在较高水平. 以 TRACE20130222 为例, 当测量分组数量为 6M 时, 各算法的 *F*1 分数最高, 且 RobustSketch、Elastic Sketch、ASketch、CF+CM 的 *F*1 分数分别为 0.95、0.89、0.97、0.95. 随着测量分组数量的递增, 各算法的 *F*1 分数逐渐降低. 当测量分组数量增加到 14M 时, 各算法 *F*1 分数分别为 0.94、0.86、0.7、0.33. 总之, RobustSketch 在流量抖动时仍能稳定取得良好的大流识别性能. 这主要归功于 RobustSketch 中的小流过滤器能在网络抖动时增加或减少循环链中的 Sketch 数量, 从而始终稳定过滤小流. 同时大流记录表能够自适应伸缩以实现大流的准确记录, 进而实现较高的大流识别性能.

4.3 大流大小识别精度

4.3.1 平均绝对误差

(1) *AAE* 与内存大小的关系

图 12 展示了不同内存大小下各算法的 *AAE* 值. 从图 12 可以看出: 在相同的内存使用量下, RobustSketch 的 *AAE* 比 Elastic Sketch、ASketch、CF+CM 更小. 以 TRACE20130222 为例, 在内存使用量从 60 KB 增加到 100 KB 的过程中, RobustSketch 的 $\log_{10}AAE$ 比 Elastic Sketch 小 2.5–2.6 倍, 比 ASketch 小 2.1–4.3 倍, 比 CM+CF 小 4.3–10.1 倍. 这主要是由于 RobustSketch 通过小流过滤思想预先排除大量小流, 使得大部分小流无法到达大流记录表, 进而降低了大流大小识别的 *AAE*.

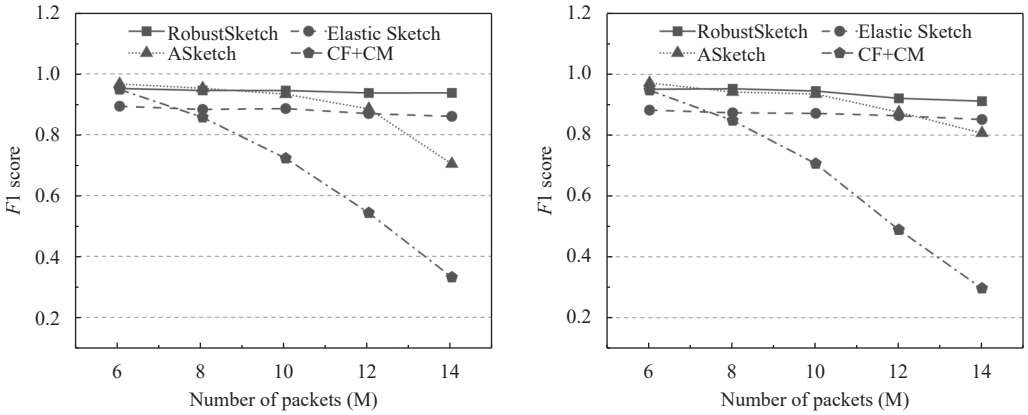


图 11 *F*1 分数与测量分组数量的关系

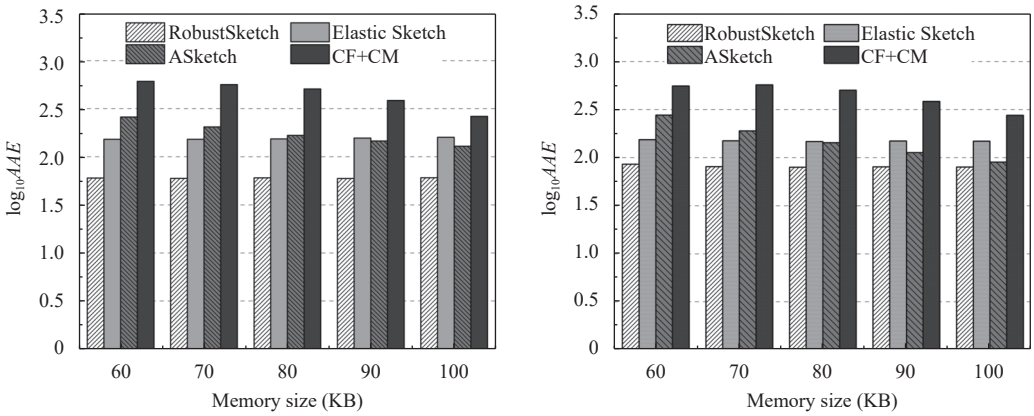


图 12 *AAE* 与内存大小的关系

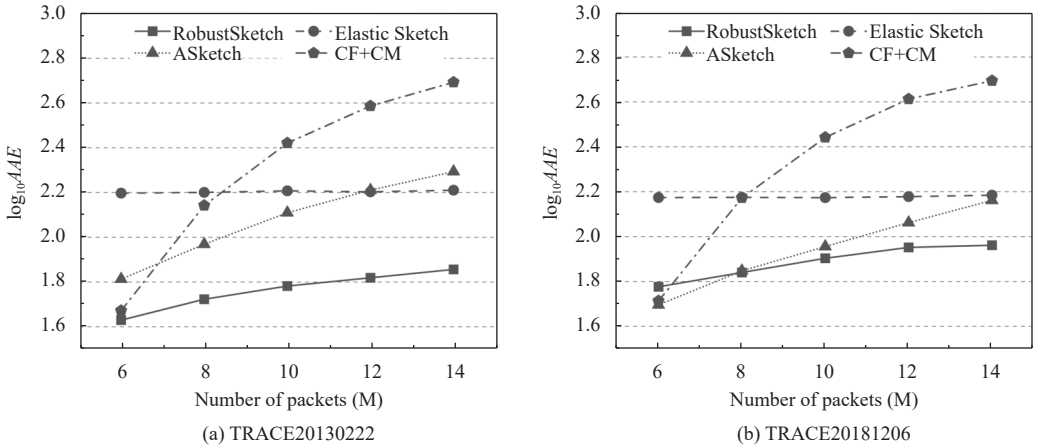
(2) AAE 与分组数量的关系

图 13 给出了不同分组数量下各算法的 *AAE* 变化情况. 从图 13 可以看出: 当内存使用量固定时, 随着测量分组数量增多, RobustSketch 的 *AAE* 始终较低. 以 TRACE20130222 为例, 在测量分组数量从 6 M 增加到 14 M 的过程中, RobustSketch 的 $\log_{10}AAE$ 比 Elastic Sketch 小 2.2–3.8 倍, 比 ASketch 小 1.5–2.7 倍, 比 CM+CF 小 1.1–6.8 倍. 总之, RobustSketch 在流量抖动时的大流识别误差最小. 这主要是归功于 RobustSketch 设计的可伸缩循环型过滤器, 可在流量抖动时持续有效的过滤小流, 减少了大量小流对大流识别的影响, 进而降低了大流大小识别的 *AAE*.

4.3.2 平均相对误差

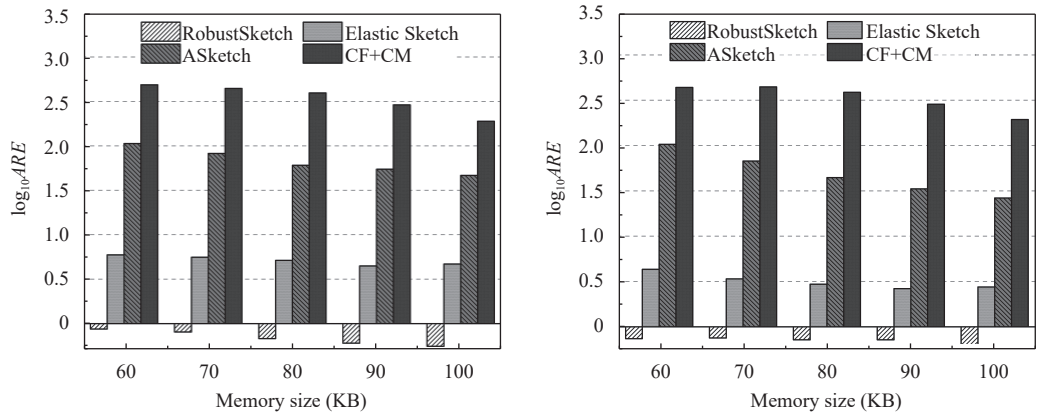
(1) *ARE* 与内存大小的关系

图 14 展示了不同内存大小下各算法的 *ARE* 值. 从图 14 可以看出: 在相同的内存使用量下, RobustSketch 的 *ARE* 比 Elastic Sketch、ASketch、CF+CM 更小. 以 TRACE20130222 为例, 当内存使用量为 60 KB 时, 各算法的 $\log_{10}ARE$ 最大, 且 RobustSketch、Elastic、HK、SS 分别约为−0.06、0.77、2.03、2.69. 随着内存使用量的递增, 各算法的 *ARE* 逐渐减小. 当内存使用量为 100 KB 时, 各算法的 $\log_{10}ARE$ 分别为−0.26、0.67、1.67、2.28. 这主要归功于 RobustSketch 持续精准的过滤掉大量小流, 并使大流无损通过过滤器, 进而通过记录表实现对大流的精确记录, 从而有效降低了大流大小识别的 *ARE*.



(a) TRACE20130222 (b) TRACE20181206

图 13 *AAE* 与测量分组数量的关系



(a) TRACE20130222 (b) TRACE20181206

图 14 *ARE* 与内存大小的关系

(2) *ARE* 与分组数量的关系

图 15 给出了不同分组数量下各算法的 *ARE* 变化情况. 从图 15 可以看出: 当内存使用量固定时, 随着测量分组数量增多, RobustSketch 的 *ARE* 明显低于其他算法, 且能够保持相对稳定. 以 TRACE20130222 为例, 当测量 6M 个分组数量时, 各算法的 $\log_{10}ARE$ 最小, 且 RobustSketch、Elastic Sketch、ASketch、CF+CM 分别约为−0.30、0.61、1.31、1.32. 随着测量分组数量的递增, 各算法的 *ARE* 逐渐增加. 当测量分组增加到 14M 时, 各算法的 $\log_{10}ARE$ 分别为−0.01、0.89、1.89、2.57. 总之, RobustSketch 流量抖动时的 *ARE* 始终保持最低. 这主要归功于两点: 1) 过滤器在流量抖动时仍能持续有效过滤小流, 减少对大流的偏低估计误差; 2) 记录表自适应增加和收缩分段, 保证了大流的完整记录, 且用概率衰减策略及时剔除了小流, 使得 RobustSketch 可以精确跟踪记录大流.

5 总 结

针对网络流量抖动导致现有基于 Sketch 的大流识别方法识别精度低的问题, 本文设计一种支持流量抖动的网络大流弹性识别方法 RobustSketch. 该方法首先设计一种循环结构的可伸缩 Sketch 过滤器筛除大量小流, 并通过适应性伸缩策略和加权平均策略保证过滤器在流量抖动时的持续有效性, 显著降低小流分组对大流识别的干扰. 然后设计一种基于动态分段哈希的可拓展大流记录表, 实现流量抖动时对大流的完整记录, 并通过概率衰减替换策略保证所记录大流的准确性, 从而实现流量抖动情况下的大流精确识别.

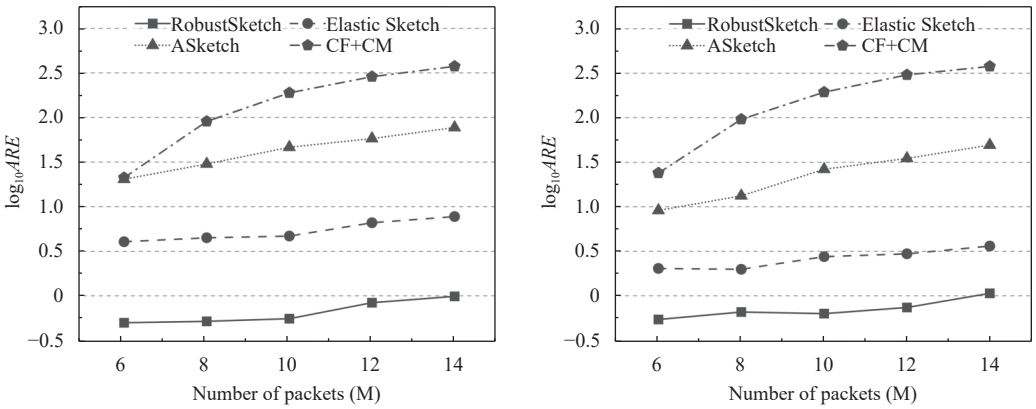


图 15 *ARE* 与测量分组数量的关系

采用真实网络流量样本进行模拟实验, 评估本文所提 RobustSketch 的大流识别精度、*F*1 分数、*AAE* 和 *ARE*. 实验结果表明: (1) 大流识别精度方面, RobustSketch 的精确率和 *F*1 分数明显优于现有方法. 当内存使用量较小时, 本文所提方法相对于现有方法的优越性尤为明显. (2) 抗抖动性方面, RobustSketch 比现有方法具有更强的适应能力. 当网络流量激增时, 本文所提方法的大流识别精确率始终保持在 99% 以上, *F*1 分数保持在 0.9 以上. 与此形成对照的是, 现有方法的精确率和 *F*1 分数均有不同程度的下降. (3) 估计误差方面, RobustSketch 的平均绝对误差和平均相对误差比现有方法更低. 当内存使用量相同时, 本文所提方法的 *AAE* 比现有方法平均小 5.8 倍, *ARE* 平均小 4.4 倍.

接下来, 将从不同的网络场景中收集更多的流量样本, 测试验证本文所提大流识别方法 RobustSketch 的有效性和适用性. 进一步, 将该方法实现并部署到各种网络应用程序中, 以帮助网络运营商进行网络管理. 未来还计划将本文所提方法应用于数据挖掘、信息检索等领域.

References:

[1] Dai HP, Shahzad M, Liu AX, Zhong YK. Finding persistent items in data streams. Proc. of the VLDB Endowment, 2016, 10(4): 289–300. [doi: 10.14778/3025111.3025112]

[2] Shi QL, Xu YC, Qi JH, Li WJ, Yang T, Xu Y, Wang Y. Cuckoo counter: Adaptive structure of counters for accurate frequency and top-k estimation. IEEE/ACM Trans. on Networking, 2023, 31(4): 1854–1869. [doi: 10.1109/TNET.2022.3232098]

[3] Xu YC, Wu WF, Zhao BH, Yang T, Zhao YK. MimoSketch: A framework to mine item frequency on multiple nodes with sketches. In: Proc. of the 29th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining. Long Beach: Association for Computing Machinery, 2023. 2838–2849. [doi: 10.1145/3580305.3599433]

[4] Shan JS, Fu YJ, Ni GQ, Luo JX, Wu ZF. Fast counting the cardinality of flows for big traffic over sliding windows. Frontiers of Computer Science, 2017, 11(1): 119–129. [doi: 10.1007/s11704-016-6053-x]

[5] Guo JR, Hong YS, Wu YH, Liu YF, Yang T, Cui B. SketchPolymer: Estimate per-item tail quantile using one sketch. In: Proc. of the 29th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining. Long Beach: Association for Computing Machinery, 2023. 590–601. [doi: 10.1145/3580305.3599505]

[6] Lall A, Sekar V, Ogihara M, Xu J, Zhang H. Data streaming algorithms for estimating entropy of network traffic. ACM SIGMETRICS Performance Evaluation Review, 2006, 34(1): 145–156. [doi: 10.1145/1140103.1140295]

[7] Jing XY, Yan Z, Han H, Pedrycz W. ExtendedSketch: Fusing network traffic for super host identification with a memory efficient sketch. IEEE Trans. on Dependable and Secure Computing, 2022, 19(6): 3913–3924. [doi: 10.1109/TDSC.2021.3111328]

[8] Zhang YD, Li JY, Lei YT, Yang T, Li ZT, Zhang G, Cui B. On-off sketch: A fast and accurate sketch on persistence. Proc. of the VLDB Endowment, 2020, 14(2): 128–140. [doi: 10.14778/3425879.3425884]

[9] Zhu JQ, Zhang K, Huang Q. A sketch algorithm to monitor high packet delay in network traffic. In: Proc. of the 5th Asia-Pacific Workshop on Networking. Shenzhen: Association for Computing Machinery, 2021. 43–49. [doi: 10.1145/3469393.3469398]

[10] Li XD, Fan ZC, Li HY, Zhong Z, Guo JR, Long S, Yang T, Cui B. SteadySketch: Finding steady flows in data streams. In: Proc. of the

31st IEEE/ACM Int'l Symp. on Quality of Service. Orlando: IEEE, 2023. 1–9. [doi: 10.1109/IWQoS57198.2023.10188743]

[11] Zhong Z, Yan S, Li ZK, Tan DC, Yang T, Cui B. BurstSketch: Finding bursts in data streams. In: Proc. of the 2021 Int'l Conf. on Management of Data. New York: Association for Computing Machinery, 2021. 2375–2383. [doi: 10.1145/3448016.3452775]

[12] Sivaraman A, Subramanian S, Alizadeh M, Chole S, Chuang ST, Agrawal A, Balakrishnan H, Edsall T, Katti S, Mckeown N. Programmable packet scheduling at line rate. In: Proc. of the 2016 ACM SIGCOMM Conf. Florianopolis: Association for Computing Machinery, 2016. 44–57. [doi: 10.1145/2934872.2934899]

[13] Curtis AR, Mogul JC, Tourrilhes J, Yalagandula P, Sharma P, Banerjee S. DevoFlow: Scaling flow management for high-performance networks. In: Proc. of the 2011 ACM Special Interest Group on Data Communication. Toronto: Association for Computing Machinery, 2011. 254–265.

[14] Zhou Y, Yang T, Jiang J, Cui B, Yu ML, Li XM, Uhlig S. Cold Filter: A meta-framework for faster and more accurate stream processing. In: Proc. of the 2018 Int'l Conf. on Management of Data. Houston: Association for Computing Machinery, 2018. 741–756. [doi: 10.1145/3183713.3183726]

[15] Roy P, Khan A, Alonso G. Augmented Sketch: Faster and more accurate stream processing. In: Proc. of the 2016 Int'l Conf. on Management of Data. San Francisco: Association for Computing Machinery, 2016. 1449–1463. [doi: 10.1145/2882903.2882948]

[16] Yang T, Zhou Y, Jin H, Chen SG, Li XM. Pyramid Sketch: A sketch framework for frequency estimation of data streams. Proc. of the VLDB Endowment, 2017, 10(11): 1442–1453. [doi: 10.14778/3137628.3137652]

[17] Basat RB, Einziger G, Mitzenmacher M, Vargaftik S. SALSA: Self-adjusting lean streaming analytics. In: Proc. of the 37th IEEE Int'l Conf. on Data Engineering. Chania: IEEE, 2021. 864–875. [doi: 10.1109/ICDE51399.2021.00080]

[18] Yang T, Jiang J, Liu P, Huang Q, Gong JZ, Zhou Y, Miao R, Li XM, Uhlig S. Elastic Sketch: Adaptive and fast network-wide measurements. In: Proc. of the 2018 Conf. of the ACM Special Interest Group on Data Communication. Budapest: Association for Computing Machinery, 2018. 561–575. [doi: 10.1145/3230543.3230544]

[19] Zhou AP, Cheng G, Guo XJ. High-Speed network traffic measurement method. Ruan Jian Xue Bao/Journal of Software, 2014, 25(1): 135–153 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4445.htm [doi: 10.13328/j.cnki.jos.004445]

[20] Tang L, Huang Q, Lee PPC. MV-Sketch: A fast and compact invertible sketch for heavy flow detection in network data streams. In: Proc. of the 2019 IEEE Conf. on Computer Communications. Paris: IEEE, 2019. 2026–2034. [doi: 10.1109/INFOCOM.2019.8737499]

[21] Yang T, Zhang HW, Li JY, Gong JZ, Uhlig S, Chen SG, Li XM. HeavyKeeper: An accurate algorithm for finding top-k elephant flows. IEEE/ACM Trans. on Networking, 2019, 27(5): 1845–1858. [doi: 10.1109/TNET.2019.2933868]

[22] Li JZ, Li ZK, Xu YF, Jiang SQ, Yang T, Cui B, Dai YF, Zhang G. WavingSketch: An unbiased and generic sketch for finding top-k items in data streams. In: Proc. of the 26th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining. New York: Association for Computing Machinery, 2020. 1574–1584. [doi: 10.1145/3394486.3403208]

[23] Ye J, Li L, Zhang WL, Chen GH, Shan YC, Li YJ, Li WH, Huang JW. UA-Sketch: An accurate approach to detect heavy flow based on uninterrupted arrival. In: Proc. of the 51st Int'l Conf. on Parallel Processing. Bordeaux: Association for Computing Machinery, 2022. 57. [doi: 10.1145/3545008.3545017]

[24] Estan C, Varghese G. New directions in traffic measurement and accounting. In: Proc. of the 2002 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications. Pittsburgh: Association for Computing Machinery, 2002. 323–336. [doi: 10.1145/964725.633056]

[25] Li YP, Wang FY, Yu X, Yang YL, Yang KC, Yang T, Ma Z, Cui B, Uhlig S. LadderFilter: Filtering infrequent items with small memory and time overhead. Proc. of the ACM on Management of Data, 2023, 1(1): 10. [doi: 10.1145/3588690]

[26] Yang T, Gong JZ, Zhang HW, Zou L, Shi L, Li XM. HeavyGuardian: Separate and guard hot items in data streams. In: Proc. of the 24th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining. London: Association for Computing Machinery, 2018. 2584–2593. [doi: 10.1145/3219819.3219978]

[27] Huang Q, Lee PPC. LD-sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams. In: Proc. of the 2014 IEEE Conf. on Computer Communications. Toronto: IEEE, 2014. 1420–1428. [doi: 10.1109/INFOCOM.2014.6848076]

[28] Cormode G, Muthukrishnan S. An improved data stream summary: The count-min sketch and its applications. Journal of Algorithms, 2005, 55(1): 58–75. [doi: 10.1016/j.jalgor.2003.12.001]

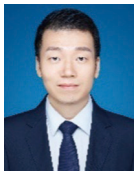
附中文参考文献:

[19] 周爱平, 程光, 郭晓军. 高速网络流量测量方法. 软件学报, 2014, 25(1): 135–153. http://www.jos.org.cn/1000-9825/4445.htm [doi: 10.13328/j.cnki.jos.004445]

熊兵(1981－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为未来网络, 网络测量, 网络安全.

刘永青(1998－), 男, 硕士生, 主要研究领域为网络测量, 数据流处理技术.

夏卓群(1977－), 男, 博士, 教授, CCF 高级会员, 主要研究领域为网络安全, 物联网, 智能电网, 智能交通.



赵宝康(1981－), 男, 博士, 副教授, CCF 杰出会员, 主要研究领域为计算机网络, 网络安全, 人工智能.

张锦(1979－), 男, 博士, 教授, CCF 杰出会员, 主要研究领域为软件工程, 人工智能.