软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(2):852−871 [doi: 10.13328/j.cnki.jos.006938]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

网络重要流检测方法综述*

钱 昊, 郑嘉琦, 陈贵海

(计算机软件新技术国家重点实验室 (南京大学), 江苏 南京 210023)

通信作者: 郑嘉琦, E-mail: jzheng@nju.edu.cn; 陈贵海, E-mail: gchen@nju.edu.cn

摘 要: 网络的管理与监测是网络领域的重要话题, 这一领域的相关技术通常也称为网络测量 (network measurement). 网络重要流检测 (network heavy hitter detection) 是网络测量的一项关键技术, 也是研究对象. 重要流指占用网络资源 (如带宽或发送的数据包数量) 超过某一给定标准的流, 检测重要流有助于快速识别网络异常, 提升网络运行效率, 但链路的高速化为其实现带来了挑战. 按出现时间顺序, 可将重要流检测方法划分为两大类: 基于传统网络框架的和基于软件定义网络 (SDN) 框架的. 围绕网络重要流检测相关的框架与算法, 系统地总结其发展过程与研究现状, 并尝试给出其未来可能的发展方向.

关键词: 网络测量; 网络重要流检测; 梗概算法; 软件定义网络; 可编程交换机

中图法分类号: TP393

中文引用格式: 钱昊, 郑嘉琦, 陈贵海. 网络重要流检测方法综述. 软件学报, 2024, 35(2): 852–871. http://www.jos.org.cn/1000-9825/6938.htm

英文引用格式: Qian H, Zheng JQ, Chen GH. Survey on Network Heavy Hitter Detection Methods. Ruan Jian Xue Bao/Journal of Software, 2024, 35(2): 852–871 (in Chinese). http://www.jos.org.cn/1000-9825/6938.htm

Survey on Network Heavy Hitter Detection Methods

QIAN Hao, ZHENG Jia-Qi, CHEN Gui-Hai

(State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

Abstract: Network management and monitoring are crucial topics in the network field, with the technologies used to achieve this being referred to as network measurement. In particular, network heavy hitter detection is an important technique of network measurement, and it is analyzed in this study. Heavy hitters are flows that exceed an established threshold in terms of occupied network resources (bandwidth or the number of packets transmitted). Detecting heavy hitters can contribute to quick anomaly detection and more efficient network operation. However, the implementation of heavy hitter detection is impacted by high-speed links. Traditional methods and software defined network (SDN)-based methods are two categories of heavy hitter detection methods that have been developed over time. This study reviews the related frameworks and algorithms, systematically summarizes the development and current status, and finally tries to predict future research directions of network heavy hitter detection.

Key words: network measurement; network heavy hitter detection; sketch algorithm; software defined network (SDN); programmable switch

网络的管理与监测是网络领域的重要话题, 这一领域的相关技术通常也称为网络测量 (network measurement). 网络测量的研究贯穿了网络技术的发展始终, 在早期的 ARPANET 中, 已有部分网络资源被专门划归于网络测量用途[1]; 随着因特网的发展, 系统化的网络测量方法与理论逐渐形成, 并不断完善[2,3]; 如今世界进入大数据时代, 网络性能不断提升, 网络测量也具有了新的价值, 正受到持续的关注. 随着分布式计算、云计算等技术的铺开, 全球的大型互联网企业, 如谷歌、微软、亚马逊、阿里等, 纷纷建立大型数据中心. 数据中心由众多以高速网络相连

* 基金项目: 国家自然科学基金 (62172206)
收稿时间: 2022-03-11; 修改时间: 2022-07-27; 采用时间: 2023-03-23; jos 在线出版时间: 2023-08-23
CNKI 网络首发时间: 2023-08-28

的服务器构成, 托管了大量的关键应用. Zhou 等人[4]指出, 随着网络技术的进步, 人们对应用性能的预期也同步增长, 此时数据中心内即使发生细微的网络异常, 如延迟抖动和丢包, 也会严重影响上层应用的性能. 尽管数据中心内的服务器和网络设备具有较高的可靠性, 但仍可能发生网络异常. 一方面, 随着规模扩张, 硬件故障几乎成为必然事件[5]; 另一方面, 分布式计算框架容易产生的 in-cast 现象[6]、网络攻击、配置错误[7]等因素都可能导致网络节点性能下降. 在上述场景下, 网络测量技术使得人们有能力在第一时间捕获网络异常的发生, 从而降低或消除网络异常带来的影响, 因此重要性凸显. 同时, 网络测量技术也对拥塞控制、路由表缓存等任务的实现起基础性作用, 使网络管理者有机会进一步优化网络的运行状态. 如今网络测量已成为网络关键功能之一, 承担支撑网络持续稳定运行的责任. 另外, 网络流量作为一种特殊的流式数据, 其测量技术在数据库[8]、数据挖掘[9]、信息安全[10]等诸多其他领域也都有应用.

网络重要流检测 (network heavy hitter detection) 是网络测量的一项重要内容, 也是本文的研究对象. 狭义的重要流指占用网络资源 (如带宽或发送的数据包数量) 超过某一给定标准的流; 广义上, 重要流检测技术也可以用于其他测量目的, 如寻找持续性最高的流 (persistent flows)、大小变化最大的流 (heavy change)、建立最多网络链接的主机 (super spreader, 也称超点) 等. 由于网络流大小分布的不均衡性[11−13], 少数流经常占用网络中的多数资源, 找出网络中的重要流成为诸多流量工程手段得以实施的前提条件. 网络重要流检测方法伴随网络测量技术的发展而进步, 本文按出现时间顺序将其划分为两大类: 基于传统网络框架的、基于软件定义网络 (SDN) 框架的. 传统网络框架下重要流检测方法的实现方式有限, SDN 的出现为其带来了更多可能性. 在链路速率不断提升的今天, 为了利用网络中设备有限的性能实现实时检测, 上述两类测量方法普遍遇到准确性与内存开销、准确性与计算开销两方面的权衡问题. 这是重要流检测方法的一大挑战, 也是各类相关工作的关注重点.

网络测量和 SDN 受到了学术界和工业界的持续关注, 在研究工作不断进展的同时, 大量文献从不同的角度出发对这些领域进行了综述. 周爱平等人[14]将网络测量方法分为抽样方法和数据流方法, 详细介绍了基于这两类方法的测量数据结构及应用. Kreutz 等人[7]将 SDN 架构分为 8 层, 自下而上逐层详细介绍了各层的研究现状. 戴冕等人[15]系统地归纳了与 SDN 结合的新型测量方法, 并从测量架构与测量对象两方面进行介绍. Tan 等人[16]详细介绍了 SDN 中带内网络遥测 (INT) 技术的现状. 尽管已有众多学者对网络测量领域进行了综述, 但这些综述均未重点关注网络重要流检测子任务. 因此, 本文围绕网络重要流检测相关的框架、算法, 系统地介绍了其发展过程与研究现状. 本文第 1 节介绍网络重要流检测问题及其实现挑战. 第 2 节分别介绍基于传统网络和基于 SDN 网络的测量框架. 第 3 节介绍基于传统网络框架的测量方法. 第 4 节介绍基于 SDN 网络框架的测量方法. 第 5 节进行总结, 并展望重要流检测的研究方向.

1 背景概述

1.1 网络测量简介

网络测量旨在通过各种可能方式与手段, 获取网络不同方面的信息, 进而允许管理人员推测网络的运行状态. 根据信息的采集方法, 可以将其分为主动测量和被动测量[17]. 根据信息采集位置, 可以将其分为基于终端节点的测量和基于网络中间节点的测量. 根据测量的对象, 可以将其分为网络结构测量、性能测量、流测量. 主动测量需要向网络内发送探测数据包, 通常在终端节点进行信息采集, 如 Ping、Traceroute 工具等; 被动测量通过某种方式捕获网络中的数据包, 然后进行分析处理, 信息采集点可以位于网络中的任何位置. 结构测量的测量对象包括网络中的拓扑和路由, 可以通过 LLDP 协议实现; 性能测量的测量对象包括时延、丢包、吞吐率等; 流测量的测量对象是网络中的数据流, 通常指传输层会话.

网络测量的一大初衷是通过测量结果优化上层网络应用的性能. 网络中上层应用之间的绝大部分数据交互通过传输层会话实现, 因此以传输层会话作为测量粒度的流测量最为贴近这一初衷, 成为研究的重点. 网络重要流检测属于流测量的一种, 这类测量任务通常采用基于网络中间节点的被动测量方案, 除重要流检测以外还包括流大

小估计、流分布估计、熵估计等任务. 网络流的一种通用标识方式使用由源主机 IP、目的主机 IP、源端口号、目的端口号、传输层协议类型组成的五元组, 这样便能唯一确定一个传输层会话. 本文中, “网络流量”指网络设备处理的所有数据包的集合, “网络流”指由上述五元组确定的网络流量的某个子集.

1.2 问题定义

网络重要流存在多种不同的定义, 本节对大量已有文献中给出的定义进行总结和分类. 给定一个时间窗口 W, 假设在此时间窗口内某网络设备 (或整个网络) 共处理了 n 个数据包, 令其处理的数据包的全体为集合 $P=\{p_1,p_2,\dots,p_n\}$. 根据网络流的五元组定义, 每个数据包 p_i 都可以划分到某个流 f_j 中, 假设 P 中的数据包共可以被划分为 m 个网络流, 将网络流按数据包计数从大到小的顺序排列, 得到网络流的集合 $F=\{f_1,f_2,\dots,f_m\}$, 假设 s_i 是网络流 f_i 的数据包计数, 则有 $s_1\geqslant s_2\geqslant\dots\geqslant s_m$. 网络重要流可以按如下方式之一定义.

● 时间窗口 W 内数据包总数超过给定阈值 θ 的网络流[11,18,19], 此时重要流集合可以表述为 $F'=\{f_i|i\leqslant j, s_j>\theta, s_{j+1}\leqslant\theta\}$.

● 时间窗口 W 内数据包占比超过给定阈值 φ 的网络流[20−24], 定义所有网络流的数据包计数之和 $N=\sum_{i=1}^{m}s_i$, 此时重要流集合可以表述为 $F'=\{f_i|i\leqslant j, s_j>\varphi N, s_{j+1}\leqslant\varphi N\}$.

● 时间窗口 W 内数据包总数排名前 k 大的网络流[11,13,20,24−29], 给定参数 k 后, 重要流集合可以表述为 $F'_k=\{f_1,f_2,\dots,f_k\}$. 使用该标准定义的网络重要流检测问题也称为 Top-k 重要流检测问题.

网络重要流检测任务要求返回时间窗口 W 内的重要流集合以及每个流对应的数据包计数 (简称流大小) 的集合. 此外, 还存在一些与重要流检测任务相关的网络测量任务, 在此一并列出.

● 重要变化检测[12,28,30,31]: 给定两个不同时间段内的网络流量数据, 汇报这两个网络流量数据中, 大小差值超过给定阈值的网络流的集合, 及其对应的差值.

● 持续流检测[28,32]: 将时间窗口 W 进一步划分为 T 个不重叠且连续的子时间窗口, 某一个流的持续度 (persistence) 定义为该流出现过的子时间窗口总数. 持续流检测要求汇报持续度高于给定阈值的流的集合. 持续流检测可以视作对时间窗口这一属性去重的重要流检测.

● 超点检测[28,33,34]: 汇报给定网络流量中, 与其他主机建立连接数量超过指定阈值的主机的集合. 超点检测通常对应网络安全方面的需求, 如检测端口扫描或 DDoS 攻击受害者, 前者的特征是单一源主机与大量目的主机建立连接, 后者相反. 广义的超点检测过程可描述为: 对网络流某一属性 (集合) A_1 的每种取值, 统计对应的另一属性 (集合) A_2 的基数 (即不同取值的计数), 并返回 A_2 基数超过某一阈值的 A_1 取值集合. 这一过程可视作首先对 $A_1\cup A_2$ 去重, 然后以 A_1 为流 ID 进行重要流检测.

1.3 挑战与研究方向

理论上, 一个准确的重要流识别算法可以分为两个阶段实现: 在分析网络流量时, 通过哈希表跟踪每一个网络流的数据包计数; 在查询重要流时, 遍历哈希表, 提取并汇报符合特定标准的重要流集合. 受制于网络环境下的约束, 上述准确算法的实时实现并不现实, 例如, 没有足够多的高速内存用于储存整个哈希表. 设计一个满足网络数据实时处理需求的算法存在若干挑战, 概括而言, 一个理想的重要流检测算法应当满足以下条件.

(1) 准确. 这包含两方面的要求: 第一, 算法应当准确地识别出所有重要流而排除那些非重要流; 第二, 算法输出的重要流大小应当尽可能接近重要流的真实大小.

(2) 计算开销低. 随着技术的进步, 速率达到 40 Gb/s 甚至 100 Gb/s 的高速链路已经在骨干网、数据中心网中占据了主流地位. 在高速链路场景下对流量进行离线存储后分析已变得不现实, 因此检测算法应当在流量到来的同时对其进行实时监测. 为了不影响网络的实时性, 检测算法处理每一个数据包的平均耗时应控制在纳秒级, 这要求检测算法对单个数据包只能进行有限的操作.

(3) 内存开销低. 为满足性能要求, 算法的总内存消耗和单个数据包的访存次数受到更为严格的限制. 计算机中的主存基于 DRAM 存储器, 这类存储器的访问时延在数十纳秒级别, 无法满足处理实时性, 故需要使用更高速

的 SRAM 存储器. CPU 的 Cache 由 SRAM 构成, 可编程交换机的数据平面也包含一定容量的 SRAM, 但这类存储的容量通常仅为数兆字节, 为了使算法能够容纳在 SRAM 中, 其数据结构需要具备极高的内存使用效率. 此外, 在基于 P4 的测量方案中, 每个数据包受到只能访问一次内存的限制.

上述几个条件在实现上存在一定程度上的互斥关系. 准确算法需要记录时间窗口 W 内网络流的所有信息, 然后进行排序, 这将消耗大量存储及计算资源. 后文介绍的近似算法牺牲了一定准确性以实现远少于准确算法的内存开销, 但对近似算法而言, 在相同内存下达成更高准确度通常意味着更为复杂的计算. 因此, 已有工作普遍在准确性与内存开销、准确性与性能方面进行了折中. 尽管如此, 更先进的测量技术能够在这几大条件之间实现更好的权衡. 早期基于抽样的测量方法[11,18,19]从原始网络流量数据中选择有代表性的子集, 通过该子集推断原始网络流量数据的特征, 极大削减了内存开销, 但引入了可观的误差. 基于数据流算法的测量方法[20,25,26]可以通过某些策略主动分离重要流和非重要流, 并使资源分配倾向于重要流, 从而在保持低内存开销的同时以更高准确度获取网络流量数据的统计信息. 进入 SDN 时代后, 控制平面与数据平面的协作使得全网资源分配更合理, 可编程的硬件资源则允许人们以极高的性能实现测量算法[12,27,30], 最终在满足性能和内存限制的前提下, 更优的测量准确度成为可能.

总之, 在网络性能不断提升的今天, 利用设备有限的性能实现准确的高速网络实时测量愈发具有挑战性. 但随着技术的发展, 网络测量正向着结果更准确、资源利用更高效的趋势发展.

1.4 评价指标

重要流检测算法通过牺牲一定准确度换取了资源开销的大幅降低, 为了量化这一权衡, 通常可以通过一系列指标对算法的准确度与资源占用进行评估. 假设某检测算法识别得到的重要流集合为 $\widehat{F'}=\left\{f'_1,f'_2,\ldots,f'_j\right\}$, 每个流对应的流大小的集合为 $\widehat{S'}=\left\{s'_1,s'_2,\ldots,s'_j\right\}$. 本节讨论了本文中用于评价算法优劣的若干指标.

首先讨论与算法准确度相关的指标, 包括评价识别准确度的召回率以及评价流大小估计准确度的平均绝对误差、平均相对误差. 在不考虑算法输出的流大小集合 $\widehat{S'}$ 的情况下, 网络重要流检测算法属于一种二分类算法, 这类算法在数据挖掘、机器学习领域已经有非常完善的评价体系. 对于真实值, 将属于重要流集合 F' 的流标记为 1, 将属于 F 但不属于 F' 的非重要流标记为 0. 对于算法输出的识别结果, 将属于 $\widehat{F'}$ 的流标记为 1, 将属于 F 但不属于 $\widehat{F'}$ 的流标记为 0. 于是, 可以得到混淆矩阵. 混淆矩阵中, TP、FN、FP、TN 分别指 true positive、false negative、false positive 和 true negative. 在一个普通的二分类问题中, 可以根据上述 4 个参数, 按如下方式定义精确率 (*Precision*)、召回率 (*Recall*)、$F1$ 指标 ($F1$ measure) 这 3 个指标.

(1) 精确率: 表示真实网络重要流占算法识别的重要流集合的百分比, 定义为:

$$Precision=\frac{TP}{TP+FP}=\frac{\left|F'_k\cap F_k\right|}{\left|F'_k\right|} \tag{1}$$

(2) 召回率: 表示真实网络重要流集合 F' 中, 算法识别出的网络重要流所占百分比, 定义为:

$$Recall=\frac{TP}{TP+FN}=\frac{\left|F'_k\cap F_k\right|}{k} \tag{2}$$

(3) $F1$ 指标: 是精确率和召回率的调和平均数, 可以综合反映以上两个指标, 定义为:

$$F1=\frac{2\cdot Precision\cdot Recall}{Precision+Recall} \tag{3}$$

若考虑算法输出的流大小集合 $\widehat{S'}$, 则还可以对算法统计得出的流大小准确度进行评价. 对于算法识别出的重要流 f'_i, 其测得大小为 s'_i, 假设其实际大小为 s_j, 则称 $\left|s'_i-s_j\right|$ 为该流的绝对测量误差, $\left|s'_i-s_j\right|/s_j$ 为该流的相对测量误差. 对算法识别出的重要流集合 $\widehat{F'}$, 定义平均绝对误差 (*AAE*) 和平均相对误差 (*ARE*) 如下.

(1) 平均绝对误差: 流集合 $\widehat{F'}$ 中所有流的绝对测量误差平均值, 定义为:

$$AAE=\frac{1}{\left|\widehat{F'}\right|}\sum_{f'_i\in\widehat{F'}}\left|s'_i-s_j\right| \tag{4}$$

(2) 平均相对误差: 流集合 $\widehat{F'}$ 中所有流的相对测量误差平均值, 定义为:

$$ARE = \frac{1}{\left|\widehat{F'}\right|}\sum_{f_i'\in\widehat{F'}}\frac{\left|s_i'-s_j\right|}{s_j} \tag{5}$$

最后考察算法的资源占用, 由于网络测量算法需要处理高速网络流量, 其消耗的资源受到严格限制. 一般而言, 可以通过吞吐率、工作集大小、处理时延等指标刻画算法的资源利用效率.

(1) 吞吐率: 重要流检测算法需要在其内部维持高性能数据结构以记录网络流量的“摘要”. 对数据结构的更新操作需要随流量到来实时完成, 而在数据结构中查询重要流信息通常是近实时或非实时的. 数据结构的更新性能通过更新吞吐率指标量化评价, 定义为每秒处理数据包数量 (packet per second, PPS), 查询性能则通过查询吞吐率量化评价, 定义为每秒查询操作执行次数 (query per second, QPS). 已有算法多重点关注更新吞吐率, 并可能使用并行计算[13]、硬件加速[12,35,36]等手段提升该指标.

(2) 工作集大小: 重要流检测算法需要频繁访问内存, 现代 CPU 中使用的高速缓存相比于 DRAM 可以提供低 1–2 个数量级的访存延迟, 但容量有限. 操作系统中进程的工作集定义为给定时段内进程频繁访问的内存, 若算法使用的工作集可容纳进 CPU 高速缓存[31], 将大幅提升其执行效率. 此外, 对于基于硬件实现的重要流检测算法, 其可用存储空间受到对应平台的严格限制.

(3) 处理时延: 网络设备中, 暂时无法处理的数据包将在缓存中排队. 高速网络的延迟会造成更高的缓存占用, 这在软件平台上将导致算法以外的内存开销, 在硬件平台上则有可能导致丢包. 由于并行计算在提高吞吐率的同时能够部分隐藏算法的处理延迟, 高吞吐算法的时延仍是一个值得关注的指标.

2 测量框架

文献 [15] 首先将网络测量方法分为测量架构和测量对象两大要素, 然后根据文献 [37] 又将测量架构分为 3 个部分: 流量的采集和预处理、流量的传输和存储、流量的分析与反馈. 在关于重要流检测方法的讨论中, 测量对象已经确定, 因此对已有工作的划分标准主要取决于测量架构. 根据本文进行的归纳, 网络重要流检测方法可以划分为以下两个层次: 底层网络框架、基于网络框架的检测算法. 其中, 网络框架决定了可行的流量采集和预处理、传输和存储方式; 检测算法基于某种网络框架, 决定了流量的分析与反馈方式. 已有的网络框架按出现时间的先后顺序可以分为传统网络框架和 SDN 网络框架, 基于传统网络框架的检测算法可按文献 [14] 的标准划分为抽样方法和数据流方法, 基于 SDN 网络框架的检测算法可划分为控制平面方法、数据平面方法和协同利用数据平面/控制平面的方法. 本节后续内容将分别介绍两种网络框架, 第 3 节、第 4 节将分别介绍基于两种网络框架的重要流检测算法.

2.1 传统网络测量框架

如第 1.1 节中所述, 重要流检测方法通常使用被动测量技术. 被动测量技术与需要向网络中发送探测数据包的主动测量技术不同, 是一种非入侵式的测量方式, 对网络本身造成的影响较小, 但需要在网络设备或链路中对途经的网络流量进行采集, 随后通过某些手段对采集的数据包进行传输与统计分析. 在传统网络测量框架中, 流量的采集与传输手段较为有限, 本文将其分为两种.

(1) 基于抓包的: 其基本思想是在网络某处设置探针采集网络流量, 然后通过网络线缆等方式将此处的网络流量发送至流量分析服务器中. 可能的探针实现包括电缆或光纤分离器、交换机的镜像端口、软件路由器操作系统中的网络接口等. 流量分析服务器中运行抓包程序, 通过 CPU 的计算能力对网络流量进行实时或离线的分析, 随后将重要流检测结果呈现给网络管理者.

(2) 基于 ASIC 的: 其基本思想是在网络设备的处理芯片中内置专有的测量功能, 在转发流量的同时直接对其进行分析, 而无需额外的网络流量传输. 该类检测手段的典型案例为思科于 1996 年引入其部分路由器产品中的 NetFlow 功能[19], 它允许用户在控制平面收集路由器处理的网络流的统计信息. 尽管基于 ASIC 的检测手段通常具有更高的性能, 但芯片本身的封闭性导致其应用场景受限.

上述两种流量的采集与处理手段都将遇到第 1.3 节中介绍的准确率、性能、内存占用几项挑战, 因此普遍借鉴流式数据处理算法的已有成果, 采用抽样方法或数据流方法以实现更好的权衡. 基于传统网络框架的重要流检测算法将在第 3 节中介绍.

需要注意的是, 尽管传统网络测量框架出现时间早于 SDN 网络测量框架, 目前基于传统框架的新检测算法仍不断出现. 此外, 大量传统网络测量框架下的算法不加修改或稍加修改后, 也可用于 SDN 网络架构下.

2.2 SDN 网络测量框架

软件定义网络 (SDN) 是网络发展历史上最大程度的架构革新之一. 在传统网络的设计中, 无全局控制器是重要原则之一[38], 这导致网络成为一个完全分布式的系统, 每个网络设备相对独立, 需要各自实现包括数据平面和控制平面在内的所有网络功能. 该设计原则使网络具有较高的开放性, 不同厂商的网络设备只需要符合一定的规范即可保证互操作性, 从而在网络发展早期为其蓬勃发展奠定了基础. 但另一方面, 该设计原则也存在众多不足. 例如, 因为网络设备的配置方式并无统一规范, 所以管理者需要针对网络设备提供的不同接口逐一进行配置, 过程低效复杂, 而网络设备中数据平面和控制平面的强耦合也导致新协议的推行受到重重阻碍[7]. SDN 为解决传统网络中存在的问题而诞生, 它将网络设备中的数据平面和控制平面分离: 底层网络设备中仅保留数据平面功能, 负责数据包的高速转发; 原本分布于每个网络设备中的控制平面功能在逻辑上合并为一个中心化的 SDN 控制平面, 负责管理处于数据平面的网络转发设备. 控制平面与数据平面之间使用称为南向接口 (southbound interface) 的标准化的协议进行通信, 如 OpenFlow[39]、ForCES[40]、OVSDB[41]等, 其中 OpenFlow 应用最为广泛. 数据平面和控制平面的分离以及两者之间标准化的通信接口使得 SDN 获得了前所未有的灵活性与开放性, 因此 SDN 同时获得了工业界和学术界的大力支持, 相关技术以极快的速度获得了落地应用.

SDN 同样影响了网络测量领域, 使得更灵活的重要流检测方法成为可能. 例如, 可以通过 OpenFlow 协议向数据平面交换机下发流表, 使其从特定端口导出网络流量的某一个子集, 而该子集可以根据管理员的测量需求与网络实际运行状态进行动态调整, 从而将有限的资源集中于重点测量对象上. 以 Open vSwitch[42]为代表的软件交换机、以 NetFPGA[43]和 P4 编程语言[44]为代表的数据平面硬件可编程技术将重要流检测方法的灵活性提升到了一个全新的高度. 前者广泛应用于多租户云服务场景下, 使得服务器也具有了 SDN 网络转发功能, 从而允许传统测量算法直接部署在转发节点上; 后者使得网络流量能在数据平面得到线速处理, 类似于第 2.1 节中介绍的基于 ASIC 芯片的方式, 不同点在于此时流量的处理方式可以通过编程指定, 对管理员可见. 大量研究[4,12,27,30,34−36]都尝试通过数据平面的可编程性实现更高效的重要流识别算法.

2.2.1 OpenFlow

OpenFlow 是 SDN 中应用最为广泛的南向接口, 于 2008 年由斯坦福大学的 McKeown 等人提出[39]. OpenFlow 最初的愿景是通过分离网络中的业务数据和实验性数据, 向研究者提供一种在现有网络中测试新型网络协议的方法, 经过多年发展, 目前已经成为 SDN 领域的基石之一. OpenFlow 的设计目标包括以下 4 点: (1) 能够以较低价格实现高性能网络架构; (2) 能够支持众多不同的科研项目; (3) 保证能够将实验性数据从生产数据中分离; (4) 不违背网络设备厂商开发私有硬件平台的需求.

一个使用 OpenFlow 协议的网络中存在两类设备: OpenFlow 交换机和控制器. OpenFlow 交换机负责网络数据转发, 内部保存了一系列流表 (flow table), 流表的每一项都由匹配规则 (match field)、动作 (action)、统计信息 (statistics) 这 3 部分组成. 对于每一个数据包, 交换机在流表中对其执行匹配, 一旦数据包与流表中的某一项匹配, 则交换机执行对应的动作、更新对应的统计信息. 匹配规则本质上是数据包头部某些域的组合, 从逻辑上定义了一个流; 动作描述了交换机对这个流执行的策略, 可能包括转发到指定端口、转发到控制器、丢弃等. 控制器使用 OpenFlow 协议与交换机通信, 执行流表的安装与更新操作. 通过在控制器上施加不同的策略, 一个 OpenFlow 交换机可以具备二层交换机、三层交换机或防火墙等不同类型设备的功能.

2.2.2 数据平面可编程与 P4 语言

OpenFlow 通过分离数据平面交换机和控制器, 为网络提供了一定的编程能力. OpenFlow 匹配规则中支持的

数据包头部域与使用的协议版本有关, 这带来了新的问题. 首先, 为了增强 OpenFlow 的通用性, 标准中需要定义更多支持匹配的数据包头部域, 仅在 1.4 版本中, OpenFlow 就已定义了 41 个头部域[44]. 其次, OpenFlow 中定义的每个头部域都需要交换机提供硬件支持, 一旦协议版本更新, 已有的交换机硬件将无法支持新的头部域. 可编程数据平面可以用于解决上述两个问题, NetFPGA 是可编程数据平面的早期方案, 它利用 FPGA 的可编程特性, 实现了一个可重用的网络硬件平台; 2014 年 McKeown 等人提出了名为 P4 的新型解决方案[44], 并迅速得到了业界关注. P4 是可编程协议无关包处理器 (programming protocol-independent packet processors) 的简称, 这是一种针对网络转发设备的编程语言, 专门用于描述网络转发设备数据平面的行为. 将编译后的 P4 程序下载到可编程的网络转发设备后, 这些设备即成为 P4 名称中描述的协议无关包处理器.

P4 编程语言的设计目标包括以下 3 点.

(1) 支持当场重新配置: 即使某一支持 P4 的交换机已经部署在了生产环境中, 控制器仍可以在其中下载新的 P4 程序, 从而修改数据包的处理逻辑. 这一设计目标极大简化了数据平面的更新过程.

(2) 协议无关: P4 交换机不同于 OpenFlow 交换机, 不与任何特定的网络协议相绑定. 通过在 P4 程序中指定数据包头部的解析方式和数据包的处理方式, 交换机即可支持某一特定的网络协议. 除了更高灵活性外, 这一设计还可以带来额外的好处: 由于可以只在 P4 程序里定义必要的协议而去除冗余的协议, 硬件资源将得到更高效的利用.

(3) 平台无关: 类似于高级语言, P4 提供了编译器将程序转换为底层平台的实际数据包处理逻辑, 从而使得同一份 P4 代码具备跨平台可移植的特性. 例如, 面向 BMv2 软件交换机时, 编译器将 P4 程序转换为 JSON 格式的配置文件; 面向 Barefoot Tofino 芯片时, 编译器则将 P4 程序转换为二进制配置文件.

P4 语言能够在可编程数据平面中实现重要流检测算法. 在语言规范中, P4 定义了 Meter、Counter、Register 等带状态的存储器操作, 使得程序可以访问交换机硬件中的 SRAM 内存, 进而实现检测算法. 所有能够被编译器成功编译到硬件平台的 P4 程序都能够以线速执行, 所以使用 P4 编写的检测算法无需考虑性能问题. 但同时, P4 也对测量算法的设计提出了特殊的要求. 例如, 交换机仅支持简单的整数算术操作, 而不能原生支持分数、指数等运算; 此外, 每个数据包在处理过程中对给定的一块内存只能访问一次, 在流水线的后一级中也无法访问前一级的内存. 尽管存在上述限制, 仍有大量研究尝试在可编程数据平面中实现重要流检测算法, 且获得了较好的效果, 相关工作将在第 4 节中介绍.

3 传统网络框架下的测量方法

在传统网络框架下, 网络流量的采集和传输手段有限. 绝大部分情况下, 需要使用某种手段将网络中的流量导出到流量分析服务器上, 通过抓包方法捕获流量, 最后通过 CPU 执行重要流检测算法. 对传统网络框架下的算法而言, 网络流量可以看作一种通用的流式数据, 网络重要流检测任务可以看作找出流式数据中出现频数最高的若干元素, 或称频繁元素. 检测频繁元素是流式数据处理领域和数据挖掘领域的重要课题之一, 除网络重要流检测外, 也可以用于其他场景, 如挖掘服务器日志信息中的频繁项、挖掘商场中的热门商品、挖掘搜索引擎中的热门关键词. 本节按文献 [14] 中的分类方式, 将重要流检测算法分为基于抽样算法和数据流算法两类, 分别进行介绍. 由于流式数据处理算法同样面临数据量大、时间空间消耗受限的问题, 传统网络框架下的重要流检测算法和数据流频繁项检测算法很大程度上可以通用, 因此本章介绍的部分算法并非针对网络流量数据而设计. 后文表 1 总结了传统网络框架下的典型重要流识别算法.

3.1 基于抽样的方法

早期, Cisco 提出的 NetFlow[19]功能基于静态抽样方法对网络流量进行采集. 在确定某一静态抽样频率 N 后, 每 N 个数据包中就有一个会被抽样, 随后算法根据抽样结果估算重要流. 由于在整个测量过程中静态抽样频率 N 不会发生改变, 这种抽样方法要么在处理低速流量时准确度不高, 要么在处理高速流量时占用过多资源. Cristian 等人[11]提出了 Sample-and-hold 算法对静态抽样进行改进, 该算法在交换机的 SRAM 中记录目前已被抽样过的流,

当遇到未被抽样的流的数据包时, 算法以某一概率进行抽样; 当遇到已被抽样过的流的数据包时, 算法 100% 进行抽样. 网络重要流包含大量数据包, 因此被算法抽样的概率高于非重要流, 这导致算法将资源偏向重要流. 经证明, Sample-and-hold 算法在相同内存下, 对重要流大小估计的误差仅为静态抽样的根号倍. Adaptive NetFlow[45]是 Estan 等人提出的另一种对 NetFlow 的优化方式, 这种方式无需手动设置抽样频率, 而可以通过网络流的特征自适应地确定抽样频率.

表 1　传统网络框架下的网络重要流检测方法

名称	分类	重要流定义	简介
NetFlow[19]	抽样方法	通用	静态抽样后进行分析
Sample-and-hold[11]	抽样方法	数据包总数超过给定阈值	静态抽样的优化, 可以在相同内存下实现更低误差
Adaptive NetFlow[45]	抽样方法	数据包总数超过给定阈值	抽样频率可以根据网络状态自适应地调整
Count Sketch[25]	数据流方法、基于Sketch	Top-k	使用Sketch保存数据流摘要, 使用堆辅助识别重要流, 具有无偏性
Count-Min Sketch[26]	数据流方法、基于Sketch	Top-k	使用Sketch保存数据流摘要, 使用堆辅助识别重要流, 误差小于CS
Conservative Update[11]	数据流方法、基于Sketch	Top-k	对CMS的优化, 提升其识别的流大小准确度
Lossy counting[23]	数据流方法、基于计数器	数据包比例超过给定阈值	把网络流量划分区间, 使用列表$\{(e, f, \Delta)\}$记录每个元素的计数, 在每个区间结束时更新列表以删除小流
Frequent[21−22]	数据流方法、基于计数器	数据包比例超过给定阈值	使用一个长度为 $1/\theta$ 的数组统计至多 $1/\theta$ 个流的大小, 作为重要流的候选集
Space-saving[20]	数据流方法、基于计数器	数据包比例超过给定阈值、Top-k	仅跟踪部分流的包计数以节省内存占用, 提出时间复杂度为O(1)的Stream summary数据结构
Unbiased Space-Saving[24]	数据流方法、基于计数器	数据包比例超过给定阈值、Top-k	概率性替换Stream summary中的元素, 实现无偏的流大小估计
HeavyGuardian[31]	数据流方法、混合方法	数据包总数超过给定阈值	基于指数衰减策略, 主动区分重要流与非重要流, 实现极高的重要流检测准确度
HeavyKeeper[13]	数据流方法、混合方法	Top-k	基于指数衰减策略, 主动区分重要流与非重要流, 实现极高的重要流检测准确度
WavingSketch[28]	数据流方法、混合方法	Top-k	灵感来源于波浪, 可以实现无偏估计, 且检测准确度远高于USS
On-off Sketch[32]	数据流方法、混合方法	持续性超过给定阈值	基于CMS进行改进, 使用标志变量限制计数器在1个时间窗口内的最大增量, 实现准确的持续流检测

3.2　基于数据流的方法

基于抽样的方法需要选择性地丢弃一部分网络流量, 通常难以获得较高的测量准确度, 因此主流的网络重要流检测多基于数据流方法. 数据流方法通常使用某种数据结构来保存已处理数据的摘要信息, 称为梗概 (Sketch). Sketch 类算法的核心思想是不完整记录数据流的所有信息, 从而降低数据结构的内存占用、提高算法的准确率. Sketch 类算法普遍使用了哈希表这一数据结构, 但作出了诸多改进. 部分学者认为 Sketch 特指基于哈希函数的数据结构. Sketch 类算法常用的数据结构除哈希表外还包括计数器, 也有部分算法设计了混合使用哈希函数与计数器的数据结构[21,34].

Count Sketch[25]是一种经典的数据流算法 (也可以代指其数据结构), 为解决数据流中的 Top-k 频繁元素检测问题而提出, 同样可以用于 Top-k 重要网络流识别. Count Sketch 寻找 Top-k 重要流网络流的整体思路类似准确算法, 但在第 1 阶段统计数据包数量时进行了改进. Count Sketch 不解决哈希冲突的问题, 而是通过使用多个哈希表来降低哈希冲突造成的误差. 其数据结构如图 1 所示, 由一个大小为 $d\times w$ 的计数器数组构成, 数组的每一行可以看作长度为 w 哈希表, 共有 d 个. 每个哈希表关联两个哈希函数, 分别是将数据包流 ID 均匀映射到 $\{1,\ldots,w\}$ 的 $h(\cdot)$,

以及将数据包流 ID 均匀映射到 $\{1,-1\}$ 的 $s(\cdot)$. 两组哈希函数内部分别要求两两独立. 数据包的插入在每一个哈希表上进行, 例如, 在第 i 个哈希表中插入属于流 f_j 的数据包时, 首先使用 $h_i(f_j)$ 将数据包定位到哈希表中的某个计数器 c, 然后将计数器更新为 $c+s_i(f_j)$. 查询流 f_i 的数据包计数时, 算法计算 $\{h_1,h_2,\dots,h_d\}$ 共 d 个哈希函数, 然后在数组中定位 d 个计数器, 随后算法遍历 d 个计数器, 确定所有计数结果的中位数, 将其作为流 f_i 的计数返回. 每次执行数据包插入后, 算法立即查询数据包对应的网络流计数, 使用一个堆维持计数值最高的 k 个网络流, 作为 Top-k 重要流输出. Count Sketch 的优势有两点: 该算法存在可数学证明的错误界、该算法给出的流计数估计无偏 (即估计值的期望等于真实值).

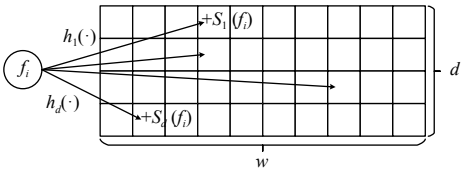


图 1 Count Sketch 数据结构

Count-Min Sketch[26]可以看作 Count Sketch 的改进算法, 同样能够解决 Top-k 重要流识别问题. 其核心数据结构与 Count Sketch 类似, 是一个大小为 $b\times w$ 的计数器数组, 但存在以下区别: 1) 对于每个哈希表, 只有一个哈希函数 $h(\cdot)$ 与之关联, 取消了 $s(\cdot)$; 2) 插入数据包时, 算法对数组中定位到的 d 个计数器执行加一操作; 3) 查询流计数时, 算法返回 d 个计数器的计数结果的最小值, 而非中位数, 这也是算法名称的由来. 该算法的设计思想基于如下原因: 哈希冲突只可能导致哈希表中某一个流的计数值变高, 而不可能导致变低, 于是对任意一个流而言, CMS 中映射到的 d 个计数器中, 最低的计数结果最接近流的真实大小, 从而具有最低误差. CMS 的应用比 Count Sketch 更广泛, 这主要是因为其具备以下优势: 1) 与 Count Sketch 相同, 存在可数学证明的错误界; 2) 尽管没有计数值无偏的性质, 可以在更小的内存下实现与 Count Sketch 相同的错误界, 或在同样内存下实现比 Count Sketch 更小的错误界; 3) Count Sketch 查询算法求中位数比 CMS 查询算法求最小值更耗时且无法并行化, 因此 CMS 性能更高.

Conservative Update[11]是 CMS 的一种优化策略, 可以降低 CMS 的流计数误差. 在 CMS 的一次插入操作中, 只有对计数结果最低的计数器的修改会影响当前网络流大小的估计, 对其他计数器的修改并不会影响对当前流的估计, 但会因为哈希冲突而在其他流的估计中引入误差. 因此, 应用 CU 策略的 CMS 在每一次插入数据包时, 仅更新计数结果最小的计数器. 该方法的不足在于, 寻找 d 个计数器的计数结果最小值的过程难以并行化, 因此 CU 策略会导致 CMS 算法性能降低.

上述 3 种基于 Count Sketch 数据结构的算法除了用于流大小测量, 也可用于流基数测量, 进而实现超点检测. Linear Counting[46]和 HyperLogLog[47,48]是常用的单流基数估计算法, 可以对数据流中不同元素的数量进行计数, 在超点检测任务中的作用可类比为重要流检测中任务的流大小估计. 因此, 使用 Linear Counting 或 HyperLogLog 算法的数据结构代替 Count Sketch 类数据结构中的计数器即可实现超点检测. Linear Counting 的数据结构是一个 b 位宽的位图 B, 初始时所有位都置为 0. 更新时, 对于每一个到来的数据包, 通过哈希函数随机选择 B 中的某一位 i 并将 $B[i]$ 置为 1; 查询时, 统计位图中 0 位的比例 V, 基数的极大似然估计即为 $\hat{s}=b\ln 1/V$. HyperLogLog 的数据结构是一个包含 2^k 个 r 位寄存器的数组 H. 更新时, 对于每一个到来的数据包, 应用哈希函数产生比特序列 ω, 计算其连续前导零的数量 $\rho(\omega)$ 并使用 ω 的低 k 位组成下标 idx, 在寄存器数组中寻址 $H[idx]$, 将其值更新为 $\max\{H[idx],\rho(\omega)\}$; 查询时, 基数的估计值为 $\hat{s}=\alpha\cdot 2^{2k}\cdot\dfrac{1}{\sum_{j=0}^{k-1}2^{-H[j]}}$, 其中 α 是修正因子.

Lossy Counting[23]可以检测贡献数据包比例超过给定阈值 θ 的网络重要流. Lossy Counting 的思想是把网络流量划分为包含固定数据包数量的若干区间, 使用由形如 (e,f,Δ) 的元素构成的列表 D 记录每个网络流 e 的计数 f, 其中 Δ 是流 e 可能的计数误差. 在每个区间结束时, 算法根据 f 和 Δ 的值更新列表以删除小流. 当用户查询重要流列表时, 算法根据 f 的值输出列表中的重要流. 该算法存在若干缺陷, 每次区间结束删除小流时应用的策略可能导致误删部分重要流, 增加计数误差; 此外, 查询重要流时算法需要遍历列表 D 以输出重要流, 引入较高计算开销.

Frequent 算法由 Karp 等人[21]和 Demaine 等人[22]同时提出, 可以检测网络流量中占总数据包比例超过给定阈值 θ 的网络重要流, 空间复杂度为 $O(1/\theta)$, 其中 θ 是 0–1 的实数. 文献 [49] 介绍了一种在数据流中查询是否有元素出现次数过半的算法, Frequent 算法可以看作这种算法的推广. Frequent 算法的核心思想是, 在给定网络流量数据中, 数据包计数占比超过 θ 的网络重要流数量不可能多于 $\lfloor 1/\theta \rfloor$ 个, 否则各网络重要流的数据包计数之和将超过网络流量数据中的数据包总数. 完整的算法共分为两趟, 第 1 趟使用一个长度为 $\lfloor 1/\theta \rfloor$ 的数组统计 $\lfloor 1/\theta \rfloor$ 个流的大小, 作为重要流的候选集; 第 2 趟对数组中的流进行准确计数, 从而得到精确的重要流集合. 第 1 趟中, 每遇到一个数据包即检查对应的流是否被数组记录, 如果已被记录则计数值加 1, 否则将数组中所有流的计数值减 1. 可以证明, 如果某网络流确实属于重要流, 则在第 1 趟的末尾其一定会被保留在数组中. 在此前提下, 考虑到网络流数据只能处理 1 次, 故算法的第 2 趟通常省略, 此时的 Frequent 算法弱化了重要流检测的要求, 仅输出 1 个保证包含所有重要流的超集.

Space-Saving[20]算法支持同时解决两种重要流识别问题, 即检测数据包比例超过给定阈值 θ 的网络重要流和检测 Top-k 网络重要流. Space-Saving 算法的核心思想是仅跟踪部分流的包计数, 使用是一个长度为 m 的表格实现, 表格的每一行都由 e_m 、 $count_m$ 、 ε_i 这 3 列构成, 类似于 Lossy Counting, $count_m$ 是流 e_m 的计数, ε_i 是流 e_m 的最大高估误差. 插入数据包时, 若其对应的网络流 f_{pkt} 已经记录在表格中, 则将该流的 *counter* 加 1; 当 f_{pkt} 尚未记录在表格中时, 分两种情况考虑: 1) 表格中记录的流数量小于 m, 此时向表格中插入 $(f_{pkt},1,0)$; 2) 表格中记录的流数量等于 m, 此时寻找表格中 *counter* 最小的流, 将其 e_m 一列替换为 f_{pkt}, *counter* 加 1, e_m 置为加 1 前的 *counter*. 该算法虽然能够较为准确地识别网络重要流, 但其更新操作使得网络流的计数包含较大的高估错.

Stream-Summary (SS) 数据结构是 Space-Saving 一文[20]的另一大贡献, 该数据结构基于链表, 逻辑上实现了 Space-Saving 核心算法中使用的表格, 优势在于时间复杂度仅为 $O(1)$. SS 对外提供的接口类似于堆, 可以在有序数据上执行插入、删除、查询容量、更新值等操作, 但效率均优于堆. SS 的底层由一个双向链表和若干循环链表组成, 如图 2 所示. 图中, 上半部分为有序双向链表, 存储表格中的列计数值; 下半部分为循环链表, 存储表格中的 e_m 和 ε_i 列. 所有值相同的计数值流都存储在同一个循环链表中, “挂靠”在双向链表中对应的节点上. 循环链表中存储计数值的元素和双向链表中存储流的元素之间需要维持相互的寻址关系, 这是通过指针实现的. SS 的插入操作与有序链表插入操作类似. 为了支持 SS 的查询操作和更新, 需要引入一个哈希表, 记录每个流在循环链表中的地址. 更新操作通过在循环链表之间移动节点实现, 例如将图 2(a) 中的 f_1 对应值加一, 结果如图 2(b) 所示. Count Sketch、CMS 等算法中的堆可以直接替换为 SS, 以达到更优的时间复杂度. SS 的缺点是引入了大量指针操作, 虽然时间复杂度仅为 $O(1)$, 但过多的内存访问次数使得该算法无法在某些硬件平台上实现.

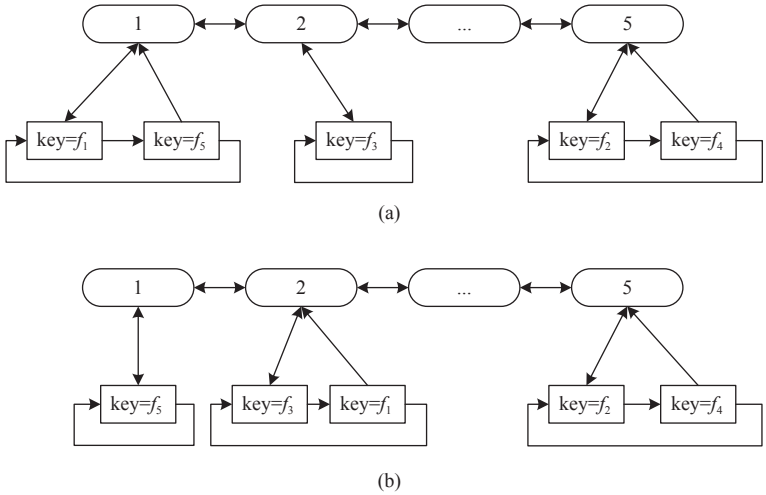


图 2 Stream-Summary 数据结构示例

Unbiased Space-Saving[24]是 Space-Saving 算法的一种变体. 该算法的核心在于概率性替换 Stream-Summary 中的元素, 以实现无偏的流大小估计. 无偏性是网络重要流检测算法的一大优良性质, 例如, 在执行多个测量结果的聚合操作时, 具备无偏性的测量算法误差不会累积. 该算法修改了 Space-Saving 的插入操作, 表格中记录的流数量等于 m 时, 假设原有最小流计数为 c, 则仅以 $1/c$ 的概率对原有流进行替换, 使得不管对原有流还是新流, 替换操作引入的误差的数学期望均为 0. 尽管 Unbiased Space-Saving 算法具备无偏性, 但其流计数仍具有较大误差.

HeavyGuardian[31]最先提出了指数衰减策略, 较准确地解决 5 类网络测量问题, 其中包括重要流检测和重大变化检测. Bucket 安排成数组形式, 每个 bucket 中有 Heavy part 和 Light part. 设置 light part 长度为 0, 使用列表记录结果; 数据包插入后立即进行查询, 如果大小等于阈值, 则流进入 heavy-hitter 列表. 数据结构里有所有流的信息, 可以进行频数估算, 进而通过差分方法实时统计频数分布, 进而实时计算熵. 根据文中的实验结果, 其准确度比先前的算法高了几个数量级.

HeavyKeeper[13]利用文献 [31] 中提出的指数衰减 (exponential decay) 策略, 专门针对网络重要流检测问题设计. 该算法可以在内存容量有限时, 实现远高于 CMS 的检测准确率和远低于 CMS 的流计数误差. HeavyKeeper 考虑了网络流的有偏特性, 这是实现高检测准确率的重要前提. 在一段包含大量网络流的网络流量数据中, 重要流在数量上只占据所有网络流的一小部分, 但却在流量上占用了大量网络资源. 与之相反, 数量上占据优势的非重要流 (或称小流) 在流量方面则只占据较小的比例. 网络数据流的这一有偏特性通常可以利用齐夫分布 (Zipf's distribution) 进行数学建模. 在 CMS 算法中, 不区分重要流与非重要流将导致前者的测量结果被后者干扰. 指数衰减策略可以有选择性地在数据结构中只保留重要流的计数, 从而实现更高的内存利用效率和测量准确率. HeavyKeeper 数据结构如图 3 所示, 从整体上看与 CMS 的数据结构类似, 都由一个 $d \times w$ 的二维数组构成, 且每一行分配一个哈希函数. 但在 HeavyKeeper 数据结构中, 数组中的元素不是简单的计数器, 而是由 (FlowID, Counter) 组成的二元组, 称为"桶". 配合指数衰减插入策略, 该数据结构的操作方式与 CMS 存在本质不同. 假设某属于流 f_i 的数据包被映射到一个桶中, 桶中现有的值用二元组 (f_j, C) 表示. HeavyKeeper 的插入算法分为以下 3 种情况讨论.

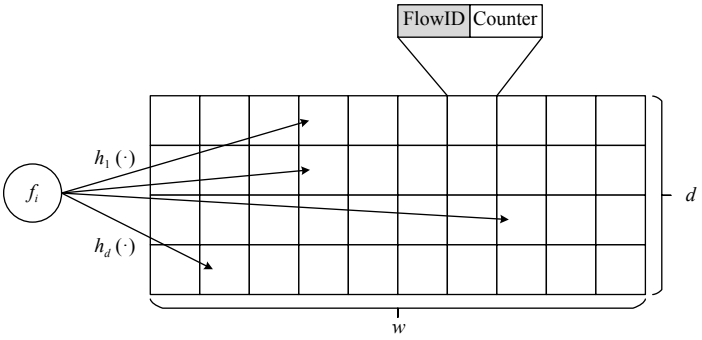


图 3 HeavyKeeper 数据结构

(1) $f_j = \text{null}$, 即桶中没有任何流: 直接将流 f_i 插入桶中, 令 $f_j = f_i$, $C = 1$.

(2) $f_j = f_i$, 即桶中的流和待插入的流是同一个: 更新桶中计数值, 令 $C = C + 1$.

(3) $f_j \neq \text{null}$ 且 $f_j \neq f_i$, 发生哈希冲突: 应用指数衰减策略, 以 1.08^{-C} 的概率令 $C = C - 1$, 否则不对桶中内容做任何修改. 如果发生了衰减, 且衰减后 $C = 0$, 则直接按第 1 种情况将流 f_i 插入桶中.

这种插入策略将导致以下行为: 重要流对应的计数器值 C 很大, 从而被衰减的概率接近于 0, 可以几乎无损地保留在数据结构中; 非重要流对应的计数器值 C 很小, 从而被衰减的概率接近于 1, 很容易被更大的流代替. 最终, 保留在每个桶中的元素都应当具有较大的流计数, 可以将这些流作为 Top-k 重要流的候选流. HeavyKeeper 的查询算法在插入阶段结束后, 遍历 $d \times w$ 二维数组中的每一个桶, 收集数据结构中记录的所有重要流的候选流, 最后输出倒序排序后的前 k 个流作为 Top-k 重要流的识别结果.

Waving Sketch[28]是一个 Top-k 重要流识别算法. Waving Sketch 在设计时注重保持无偏的特性, 不引入累计误

差, 且可以实现比先前的无偏算法[24,25]更高的准确度. Waving Sketch 算法的核心数据结构是一个长度为 l 的数组 B, 数组的每一个元素称为“桶”, 包含计数器部分和重要流部分. 算法的一部分思路借鉴于 Count Sketch, 对于一个新到来的数据包, 算法首先通过哈希函数 $h(\cdot)$ 将其映射到数组中的某一个桶中, 然后通过哈希函数 $s(\cdot)$ 将其映射到 $\{1,-1\}$ 中, 作为计数器的增量. 每次插入数据包 p 后, 通过 $s(p)\times B[h(p)].counter$ 估计其所在流的计数, 并通过桶中的重要流部分缓存计数值最大的部分流. 该设计保证了流计数始终无偏, 而重要流因为被缓存于重要流部分, 其准确度也得到保障. 此外, Waving Sketch 提出, 能够实现 Top-k 重要流检测的算法经过不同的配置, 可以解决以下 3 类问题: Top-k 重大变化检测、Top-k 持续流检测、Top-k 超点检测, 配置方法如表 2 所示.

表 2　解决不同检测任务时 Waving Sketch 的配置

检测任务	输入	是否去除重复元素	汇报内容
重要流	流ID	否	数据包计数最大的 k 个流
重要变化	流ID	否	数据包计数变化最大的 k 个流
持续流	流ID	是	数据包计数最大的 k 个流
超点	<源IP, 目的IP对>	是	计数最大的 k 个源IP

On-off Sketch[32]可以准确地解决重要流检测问题的变种—持续流检测问题. 文献 [28] 中提出了一种使重要流识别算法支持持续流检测的通用方式, 即为原有算法前置一个 Bloom Filter[50], 使得每个流的计数在每个时间窗口内至多增加 1. Bloom Filter 存在假阴性现象, 而大部分重要流检测算法存在假阳性现象, 因此该通用方法误差界通常难以分析. 例如, 将该方法应用于 CMS 时, 流的持续度通常会被严重高估. On-off Sketch 基于 CMS 进行了优化, 其核心思想是: 在某一时间窗口内, 任意网络流的持续度最多增加 1, 反映到 CMS 数据结构中, 每一个计数器的最大增量也为 1. 该算法为 CMS 的每个计数器附加了一个标志域, 将其在每个时间窗口内的最大增量限制到 1, 从而大幅提升持续流检测准确度.

此外, 在更广义的流式数据处理范畴, 也有学者对测量方法的分布式部署进行了研究. Mergeable Summary[51]一文对分布式环境下的数据流测量算法进行了理论分析, 总结出了其需要满足高效、可归并的特性. Topkapi[52]一文提出了一种基于 Count-Min Sketch[26]和 Frequent Algorithm[21,22]的改进型算法, 可以利用分布式环境下的并行处理能力提升性能.

4　SDN 框架下的测量方法

SDN 对传统网络架构进行了重要变革, 将网络划分为控制平面与数据平面[7]. 经过多年发展, 目前 SDN 架构下的控制平面[39]与数据平面[44]均具备可编程能力, 这为网络重要流检测提供了更多实现手段. 表 3 总结了 SDN 框架下的典型网络重要流检测方法. 根据对两大平面的利用程度不同, 本文将 SDN 架构下的网络重要流检测方法划分为控制平面方法、数据平面方法和协同利用数据平面/控制平面的方法. 控制平面方法主要利用控制器收集全网信息及下发配置的能力, 提升传统重要流检测方法的效果, 早期工作多集中于此类方法. 例如, 可以在控制器上利用 OpenFlow 协议与数据平面交换机交互, 从而实现精细化的流量导出. 随着数据平面可编程能力的提升, 业界逐渐弃用先抓包后分析的范式, 转而在数据平面处理流量的同时执行检测算法, 提出了众多数据平面方法. 此类方法的特点是将实时流量分析安排在数据平面, 并可选地利用控制平面辅助解算测量结果. 同时, 部分学者正在探索协同利用控制平面与数据平面的重要流检测方法, 此类方法可以结合上述两种方法的优点, 兼顾控制平面的全网协调能力和数据平面的高速处理能力, 提升整体资源利用效率, 具有较大潜力. 通过分析相关文献资料, 本文将 SDN 框架下重要流检测方法的研究方向归纳为以下几点.

(1) 将已有传统网络框架下的重要流检测方法实现在 SDN 可编程数据平面下. SDN 数据平面具有多种可能的形态, 包括以 Open vSwitch 为代表的软件交换机和以 P4、NetFPGA 为代表的可编程硬件平台. 以云计算多租户环境为例, 服务提供商运用 SDN 架构为客户提供功能灵活的虚拟网络服务, 这类服务通常超出传统交换机的处

理能力, 因此服务提供商普遍使用运行软件交换机的通用 x86 或 ARM 服务器代替部分硬件交换机, 实现包括云网关、宿主机虚拟交换在内的网络功能[53,54]. 通用软件交换机的引入使得传统重要流检测算法可以直接迁移到数据平面而无需任何修改. 另一方面, 软件交换机中 CPU 的数据处理能力远低于支持 P4 语言的硬件可编程交换机, 因此将已有重要流检测方法移植到 P4 平台也成为研究方向之一.

(2) 设计适用于可编程数据平面硬件的全新重要流检测方法. 可编程数据平面的专用硬件对算法的实现提出了特殊的要求, 其特殊的数据包处理方式与内存访问模式使得只有经过特殊设计的算法才能运行. 例如, 原生的 P4 语言中不支持浮点数和对应的运算, 算法需要能够通过匹配-动作流水线的编程模型进行表达, 且流水线靠后阶段也无法访问分配至靠前阶段的内存. 尽管部分传统框架下的重要流检测方法经适配可以运行在 P4 交换机上, 针对硬件特性设计全新检测方法将能够带来更大的性能提升.

(3) 控制平面与数据平面协作, 提升重要流检测方法的效果. 控制平面虽不负责高速数据处理, 但也具备一定的计算能力. 当算法仅部分实现在数据平面中时, 其数据平面部分通常可以起到降低算法控制平面部分复杂度、提升算法整体检测效果的作用. 此外, 控制平面具有全网视野良好的优势, 利用控制平面调度网络中多个测量节点任务分配, 最终收集、合并全网测量数据, 同样有机会提升测量效果.

表 3　SDN 框架下的网络重要流检测方法

名称	分类	重要流定义	简介
OpenSample[55]	控制面方法 (基于抽样)	数据包总数超过给定阈值	基于SDN架构的抽样方法, 利用TCP的序列号信息实现低延迟的重要流检测
CloudSentry[53]	控制面方法 (软件交换机)	Top-*k*	在多租户云服务环境的软件网关上, 实时监控CPU占用率, 只在突发CPU占用时分析重要流, 并可以实现租户级的流量限制
OpenSketch[27]	数据面方法 (NetFPGA)	Top-*k*	将网络测量算法抽象为若干数据平面原语的组合, 数据平面基于原语进行编码, 控制平面负责解码
FlowRadar[29]	数据面方法 (P4)	Top-*k*	数据平面使用异或操作对网络流量进行编码, 控制平面分单机和全网两阶段进行解码
HashPipe[35]	数据面方法 (P4)	Top-*k*	Space-Saving在P4交换机中的近似实现. 在多级流水线中部署哈希表, 在重要流表的子集中查找最小流以代替全表最小流
Elastic Sketch[12]	数据面方法 (多种硬件平台)	数据包总数超过给定阈值、重大变化	灵感来源于陶片放逐法, 基于投票模型确定大小流的归属, 用不同数据结构维持大小流的计数
BeauCoup[56]	数据面方法 (P4)	数据包总数超过给定阈值	灵感来源于奖券收集问题, 可以在数据平面利用Match-Action table同时执行大量符合要求的检测任务
Elastic Trie[34]	数据面方法 (FPGA+P4仿真)	数据包比例超过给定阈值、重大变化、超点	完全基于数据平面维护一个IP前缀树结构, 在每个节点中统计对应子网的流量, 在流量超过阈值时主动向控制面推送重要流信息
CocoSketch[36]	数据面方法 (P4)	数据包总数超过给定阈值	Unbiased Space-Saving在P4交换机中的近似实现. 解决了硬件中内存访问和循环依赖的问题, 可以实现任意部分键上的流大小查询
UnivMon[30]	控制面/数据面协作 (P4)	数据包比例超过给定阈值、重大变化、超点	基于Universal streaming理论, 在数据平面使用多个Count Sketch进行分层抽样, 同时解决多种网络测量任务
Network-wide HH detection[57]	控制面/数据面协作 (P4)	数据包总数超过给定阈值	利用P4解决CDM问题, 为每个交换机设置自适应的局部阈值, 仅在流可能是重要流时触发全局统计, 大幅降低通讯开销
OmniMon[33]	控制面/数据面协作 (P4)	超点	重构终端主机、交换机、控制器在执行检测任务时的分工, 利用各自的优势协作完成检测任务, 实现高效准确的重要流检测

4.1　控制平面或数据平面方法

目前 SDN 框架下的重要流检测方法中, 仅利用控制面能力的工作相对较少, OpenSample[55]是其中的典型方

法. OpenSample 是一个基于抽样的测量平台, 已有的抽样方法中, 低抽样频率将导致高误差, 高抽样频率又会导致高资源消耗, 总体效果不如数据流方法. OpenSample 观察到, 数据中心内的流量大部分都是 TCP 流量, 而 TCP 数据包的头部包含序列号, 同一条流两次采样之间序列号的差可以用于准确估算其传输的数据量, 将数据量与采样时间差相除即可得到该流单位时间内的流大小. 由于利用了序列号信息, 该方法在短时间内即可得到良好的采样准确度, 应用在 SDN 流量工程中时, 控制循环的周期从先前方法的 1–5 s 降低至了 100 ms. 典型的应用场景包括及时更新 SDN 交换机的表项, 使重要流绕开拥塞链路.

基于 x86 或 ARM 通用服务器、利用虚拟化技术实现的软件交换机在云计算多租户环境下存在广泛应用, 典型案例如具备 VXLAN 多租户隔离功能的云网关. 软件交换机具有高灵活性, 但性能弱于硬件交换机, 当负载升高时更有可能产生高延迟或丢包. 传统框架下的重要流检测方法可以直接应用于软件交换机, 但这类算法会进一步加重软件交换机的 CPU 负载. CloudSentry[53]观察到突发的 CPU 占用与重要流存在相关性, 因此提出了一种两阶段的监控框架: 持续运行一个轻量化服务监控交换机 CPU 占用率, 仅当观测到突发 CPU 占用时导出数据包并执行重要流检测. 根据重要流检测结果, CloudSentry 还能够定位产生问题的租户或服务, 即时进行反馈以限制其数据包发送速率.

大量工作尝试引入硬件可编程数据平面的高速数据处理能力, OpenSketch[27]是最早探索该方向的工作之一. OpenSketch 本质上是一个通用的网络测量框架, 可以支持包括重要流识别、超点识别在内的多项网络测量任务. OpenSketch 并未提出新的测量算法, 而是给出了一种在可编程数据平面中表达已有测量算法的方式, 类似编程语言中“函数库”的概念. OpenSketch 将数据平面抽象为 Hashing、Classification、Counting 这 3 个阶段, 将不同的算法需要的操作抽象为一族原语, 管理员使用原语实现不同的测量任务, 控制器将其配置到数据平面中; 数据平面中的各原语对网络流量进行统计, 随后将统计结果发送至控制平面, 由控制平面解算重要流识别等测量任务的实际结果. OpenSketch 的一大贡献是提出了一种重要流检测数据平面方法中常用的数据处理模式: 数据平面使用硬件对网络流量进行高效编码、控制平面解码数据平面的编码结果, 该模式被后续众多数据平面算法采纳. 尽管在这种模式中控制平面与数据平面进行了一定程度的协作, 但控制平面的工作仅局限于求解数据平面编码, 而不涉及 SDN 架构下的全局调度, 因此本文仍将此模式归类为数据平面方法. OpenSketch 的缺点在于, 尽管支持同时进行多个测量任务, 但对数据平面有限资源的划分可能导致测量准确度的下降.

当需要在硬件可编程数据平面实现重要流检测算法时, 硬件适配问题成为一大突出挑战. 以 P4 交换机为代表的可编程数据平面通过牺牲一定通用性换取了高性能, 因此在传统 CPU 上可行的算法未必能够直接迁移到硬件平台. 为了解决该问题, 典型的思路可以分为: 1) 改进已有算法, 使其可以运行在硬件平台, 如 HashPipe[35]; 2) 设计适应硬件平台的全新算法, 如 FlowRadar[29]、BeauCoup[56].

Space-Saving[20]是传统网络框架下的经典 Top-k 重要流检测算法, 但作者提出的 Stream Summary 数据结构虽然时间复杂度低, 却因为大量的内存访问操作无法直接在数据平面实现. HashPipe[35]对 Sapce-Saving 进行了改进, 在交换机的多级流水线上部署哈希表, 实现对 Space-Saving 的近似. Space-Saving 通过维护一个容量为 k 的表格记录 Top-k 重要流及各自的流大小, 当表格容量满时, 新流将会替换表格中现有流大小最小的流. P4 交换机在每个 Stage 中只能读写一次表格, 因此实现 Space-Saving 的关键挑战在于寻找表中最小流. 在 HashPipe 中, 作者将表格拆分为 d 个哈希表, 部署至 d 个 Stage 中. 数据包在每个 Stage 索引表中的一个流, 最终取 d 条流中的最小流作为全表最小流的近似.

FlowRadar[29]提出了一种数据中心场景下对 NetFlow 的改进, 基于数据平面编码、控制平面解码的测量模式, 可以实现对网络中所有流大小的监测. FlowRadar 的数据平面通过异或操作对流经的网络流进行编码, 控制平面收集网络中所有交换机的测量结果后分单机和全网两步解算所有网络流的大小. 在网络流大小的解算过程中, 可以联合使用堆等数据结构以实现 Top-k 重要流检测. FlowRadar 的缺点在于, 控制平面的解码算法复杂度较高, 且不一定能够保证解算成功.

Elastic Sketch[12]是一个通用的网络测量算法, 可以使用单一数据结构解决包括重要流检测在内的 6 种网络测量任务. Elastic Sketch 的通用性不只局限于测量任务, 还包括实现手段, 其支持的平台包括 CPU、FPGA、GPU、

P4 等. Elastic Sketch 与文献 [13,31] 类似, 同样采用分离大小流的策略来提升准确度. 算法的灵感来源于古希腊的陶片放逐法, 某个流在首次插入数据结构时储存于基于数组的大流部分, 属于该流的数据包为该流投赞同票, 其余数据包为该流投反对票. 当反对票数量超过赞同票的一定倍数时, 认定该流成为小流, 被逐出到基于 CMS 的小流部分. 该算法的处理速度优于 UnivMon, 同时取得了更高的准确率.

BeauCoup[56]是一种针对可编程数据平面设计的重要流检测方法. 作者将包括网络重要流检测在内的一类网络测量任务总结为如下形式: 汇报数据包某个属性的不同值数量超过指定阈值的流. 例如, 重要流检测任务可以表达为汇报具有不同时间戳的数据包总数超过给定阈值的流, 超点检测也可以使用类似形式表达. BeauCoup 的设计来源于奖券收集问题 (coupon collector), 简化表述为: 给定一组指定数量的奖券, 进行带放回的均匀抽取, 平均需要多少抽可以集齐所有奖券. BeauCoup 将每个流的数据包映射为奖券编号, 将数据包到来的过程抽象为抽取奖券的过程. 给定一个重要流阈值, 将其作为奖券收集问题的答案, 反解所需奖券数量, 即可在概率上实现对应阈值的重要流检测. BeauCoup 的一大贡献是利用 P4 的 Match-Action table 抽象, 在数据平面完整实现了上述奖券收集问题. 此外, 只要符合 BeauCoup 要求的形式, 该算法可以同时执行大量参数不同的重要流检测任务.

Elastic Trie[34]是一种利用 IP 前缀树, 完全在数据平面进行重要流检测的方法. IP 网络中的地址划分具有层次化结构, 因此普遍应用前缀树实现最长前缀匹配路由查找. 前缀树每一层的节点对应一个特定大小的子网, 通常而言根节点对应整个 IP 网络, 底层叶节点对应一个 IP. Elastic Trie 不在前缀树中存储路由信息, 而是存储对应子网的流量计数. 数据结构初始化时, 仅包含根节点, 代表整个网络; 随着流量的到来, Elastic Trie 根据流量特征的变化动态调整前缀树结构, 即扩展树中的大流量叶节点成为子树, 收缩小流量子树成为叶节点. 当某一节点的流量超过阈值时, 该算法即会向控制面产生一个重要流报告并重置节点计数. 与先前的方法比较, Elastic Trie 具有以下优势: 1) 完全在数据平面检测重要流并上报, 控制平面仅负责收集信息; 2) 数据平面能够实时检测重要流, 对异常流量有较快的响应速度; 3) 在传统单一重要流检测之外, 还支持超点检测和层次化重要流检测, 后者指检测产生大量流量的子网. 本方法需要在数据平面修改前缀树的内容, 因此无法直接使用 P4 提供的最长前缀匹配 (LPM) 功能. 作者使用寄存器和哈希表资源构建了 LPM 的代替, 并在软件交换机上成功进行了 P4 仿真.

CocoSketch[36]也尝试在数据平面实现层次化重要流检测, 并沿用了 Sketch 类算法的思想. 作者指出此类问题可以归结为更广义的子集和 (subset sum) 问题: 当需要测量某个流量集合的总流量大小时, 可首先利用已有方法测量每条单流的大小, 并对集合中流的大小求和. 然而, 这一思想仅适用于那些测量误差的方差较小的方法, 如 USS[24], 否则会因误差累积而失效. CocoSketch 在 P4 平台上近似实现了 USS 算法. 为限制内存访问次数, CocoSketch 提出随机方差最小化策略, 仅选中流大小表的一个子集进行 USS 更新操作; 为避免内存访问中的循环依赖, CocoSketch 将流大小表拆分为多个独立实例, 每个实例仅在一个表项上执行随机方差最小化, 最终在控制平面取每个实例测量值的中值以降低误差. 实验结果表明, CocoSketch 的硬件优化仅导致了小于 10% 的精度损失.

4.2 控制平面/数据平面协作方法

如今, 数据中心网络正变得愈发多样化, SDN 技术与各类虚拟化技术使得传统网络设备被 P4 交换机、软件网关、虚拟交换机等部件代替, 这些网络部件具有更灵活的数据处理能力以及更强大的配置接口, 通过合理的统一控制, 这些部件将不再独立工作, 而成为紧密联系的整体. 在此背景下, 网络重要流检测方法同样需要把网络作为一个整体看待, 结合控制平面与数据平面各自的优势, 合理利用各网络部件的能力, 实现更优的测量效果.

多层负载均衡是数据中心提升流量处理能力的典型架构. 例如, ECMP 技术可以将流量分摊在多条等价路由上, 数据包可能通过多个不同的边缘交换机进入目标网络. 在此背景下, 必须使用分布式的全网协同方法进行网络重要流检测, 否则某重要流可能因为在所有边缘交换机上都未达到阈值而被错误地忽略. Harrison 等人[57]将该问题抽象为 CDM (continuous distributed monitoring) 问题, 假定网络存在 n 个边缘交换机, 每个进入网络的数据包经过且只经过其中之一, 并只在经过的交换机上产生流量记录; 存在一个中央控制器, 能够在每台交换机上独立设置每条流的上报阈值; 当交换机检测到某条流大小超出阈值, 即上报流大小, 并由控制器统计流在全网范围内的总大小. 为解决 CDM 问题, 作者将每台交换机的流上报阈值设置为全局重要流阈值的 $1/n$, 控制器不断收集交换机上

报的数据, 并保守假定未上报数据的流大小为其上报阈值减一. 当某条流的总流量可能超出重要流阈值时, 控制器主动查询该流在所有交换机上的统计信息, 若确实为重要流, 则重置流计数并使用指数加权移动平均更新其在每个交换机上的上报阈值. 该方法通过数据平面主动触发上报, 大幅降低了控制平面的通信开销. 此外该方法也保留了在数据平面上实现高性能流大小估计算法的可能性.

UnivMon[30]尝试在数据平面下利用单一数据结构记录通用流信息, 并在控制平面解算以解决多种重要流检测问题, 此外控制平面还负责协调全网范围内的测量, 实现 One-big-switch 抽象. UnivMon 算法得名于其支持通用检测 (universal monitor), 可以实现的任务包括检测数据包占比超过给定阈值的流、检测发生重大变化的流、检测超点等. UnivMon 不将数据平面资源划分给多个检测任务使用, 因此当检测任务复杂度增加时 UnivMon 可以获得优于 OpenSketch 的效果. UnivMon 的数据平面基于 universal streaming 理论, 原始数据流经过多层采样产生多条子流, 每条子流包含的数据包数量都为上一条子流的一半; 每条子流记录在对应的 Count Sketch 中, 流与流之间并行处理; 同时, 数据平面需要为每条子流维持重要流集合. 控制平面通过 Universal Sketch 性质对数据平面的结果进行解码, 在应用不同 g 函数时, 可以求解不同测量任务所需结果. 针对全网测量, 控制平面通过求解优化问题向各交换机下发测量配置文件, 使得每台交换机只关注网络流的特定子集. 尽管 UnivMon 在单一数据结构中实现了通用检测功能, 但其需要同时操作多个 Count Sketch 数据结构、计算多达数十个哈希函数, 在实际使用中存在性能问题. 另一方面, 该算法给出的误差界较为宽松, 重要流检测误差较大.

在单一节点的网络测量方法中, 高准确度与低资源消耗不可兼得. OmniMon[33]指出, 当测量范围扩展到全网时, 更多数量的节点可以使人们绕过单一节点的资源限制, 实现更好的准确度-资源消耗折中. 为此, OmniMon 使用分-合两个步骤重构了网络测量框架: 首先, 将网络测量任务划分为多块子任务, 在网络中的不同实体间调度; 然后, 由控制平面对全网资源进行管理, 结合端系统与可编程交换机的优点, 在网络中无错误地跟踪每一个流, 最终实现包括超点检测在内的 11 种网络异常检测. 为此, OmniMon 提供了一致性和可解性两方面的保证: 通过一个全网范围内的纪元同步机制, 使网络中地实体在多数时间处于同一个纪元, 并只在边界产生时间偏离; 通过为每个交换机和每条流建立线性方程组, 保证在数据中心网络情况下存在唯一解. 作者分别在控制器、端系统侧和 P4 交换机侧实现了 OmniMon 系统, 并取得了良好的实验效果.

5 总结与展望

本文对网络重要流检测方法进行了较为详细的调研与综述. 首先, 给出网络重要流检测问题的提出背景, 总结已有文献中对网络重要流的不同定义, 讨论网络重要流检测算法需要满足的若干条件及实现挑战, 并给出这类算法的通用评价指标体系. 接着, 综合已有文献中的分类方式和网络重要流检测的实际情况, 将本文的讨论划分为底层网络框架、基于网络框架的检测算法两部分. 在网络测量领域, 底层网络框架按时间发展可分为传统框架和 SDN 框架, 前者具有较大限制, 而后者的出现为网络测量注入了新的活力, 本文概述了两类框架包含的各主要方面. 检测算法基于底层的网络框架, 同样可以分为两类. 于是本文在最后逐一介绍了两类网络框架下具有代表性的网络算法, 总结了各算法的优劣所在. 根据对已有方案及领域最新进展的梳理, 本文在此对网络重要流检测领域的发展方向进行展望, 总结为以下几点.

(1) 在传统网络框架下, 仍有机会继续提升重要流检测准确度. 目前 Sketch 类数据结构、大小流分离思想已广泛应用于重要流检测, 但仍可能通过引入新颖的测量理论进一步提升准确度. Huang 等人提出的 NZE Sketch[58]可视作该方向的尝试, 作者利用最优化与压缩感知方法在 SDN 下实现了对几乎所有流大小的准确测量, 但存在一定性能问题, 且无法直接用于重要流检测. 未来仍需在已有工作的基础上探索更多测量机制的应用, 寻求性能优秀且准确度更高的测量算法.

(2) 增强测量方法对网络环境的适应性. 研究指出, 在真实的网络环境下, 网络流量呈现的特征并不稳定. 已有算法大多使用一组固定的超参数初始化数据结构, 无法保证其在多变的网络流量特征下始终最优. 通过设计更灵活的数据结构以及能够响应网络流量特征变化的反馈机制, 将有机会使算法在更细的时间粒度内保持高效, 提升

整体资源利用效率.

(3) 探索或优化适用于广义重要流检测任务的方法. 如第 1.2 节所述, 广义的网络重要流检测还包括了重要变化检测、持续流检测、超点检测等任务, 这些任务与狭义重要流检测有关, 但各自具有不同的实现挑战. 例如, 超点检测需要能够识别子流并进行去重, 目前主流方法借助基数测量数据结构实现, 但总体上缺乏更深入的研究.

(4) 引入新的软硬件技术. 目前, SDN 架构下出现的数据/控制平面分离思想, 软件交换机、可编程硬件交换机等技术已大幅改变了传统网络架构, 但网络的演进并未停止. 目前各类新型软硬件技术正不断出现, 例如, XDP (express data path)[59]为 Linux 内核网络提供了灵活且高性能的旁路机制; 以智能网卡为代表的 DPU (data processing unit) 融合了传统网卡、FPGA、嵌入式 ARM 系统等硬件资源, 为主机提供多样化的硬件卸载能力; 广泛使用的可编程交换芯片 Tofino 已推出了处理能力更强的第二代和第三代芯片; 学界也已出现通过增加旁路硬件增强已有交换芯片的尝试[60]. 上述这些网络架构变化为网络重要流检测提供了新的资源, 也提出了新的需求, 因此目前亟需提出适应网络架构演进的测量算法.

(5) 完善 SDN 环境下的测量任务全局协调能力. 在 SDN 环境下, 网络控制器天生具有全局测量任务分配与结果收集的能力. 随着网络中软硬件资源多样性的提升, 网络重要流检测已经从典型的单机流处理问题演变为分布式异构硬件下的流处理问题, 这也对控制器的任务调度、异构硬件资源协调使用能力提出了更高的需求.

References:

[1] Vinton GC. Formation of Network Measurement Group (NMG). IETF RFC 323, 1972. [doi: 10.17487/RFC0323]

[2] Leland WE, Taqqu MS, Willinger W, Wilson DV. On the self-similar nature of Ethernet traffic. ACM SIGCOMM Computer Communication Review, 1993, 23(4): 183–193. [doi: 10.1145/167954.166255]

[3] Vern E. Measurements and analysis of end-to-end Internet dynamics [Ph.D. Thesis]. Berkeley: University of California, 1997.

[4] Zhou Y, Sun C, Liu HH, Miao R, Bai S, Li B, Zheng ZL, Zhu LJ, Shen Z, Xi YQ, Zhang PC, Cai D, Zhang M, Xu MW. Flow event telemetry on programmable data plane. In: Proc. of the 2020 Annual Conf. of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication. ACM, 2020. 76–89. [doi: 10.1145/3387514.3406214]

[5] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. Communications of the ACM, 2008, 51(1): 107–113. [doi: 10.1145/1327452.1327492]

[6] Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: Cluster computing with working sets. In: Proc. of the 2nd USENIX Conf. on Hot Topics in Cloud Computing. Boston: USENIX Association, 2010. 1–7 [doi: 10.5555/1863103.1863113]

[7] Kreutz D, Ramos FMV, Veríssimo PE, Rothenberg CE, Azodolmolky S, Uhlig S. Software-defined networking: A comprehensive survey. Proc. of the IEEE, 2015, 103(1): 14–76. [doi: 10.1109/JPROC.2014.2371999]

[8] Soliman MA, Ilyas IF, Chang KCC. Top-*k* query processing in uncertain databases. In: Proc. of the 23rd IEEE Int'l Conf. on Data Engineering. Istanbul: IEEE, 2007. 896–905. [doi: 10.1109/ICDE.2007.367935]

[9] Mirylenka K, Cormode G, Palpanas T, Srivastava D. Conditional heavy hitters: Detecting interesting correlations in data streams. The VLDB Journal, 2015, 24(3): 395–414. [doi: 10.1007/s00778-015-0382-5]

[10] Zhang Y, Fang BX, Zhang YZ. Identifying heavy hitters in high-speed network monitoring. Science China Information Sciences, 2010, 53(3): 659–676. [doi: 10.1007/s11432-010-0053-5]

[11] Estan C, Varghese G. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. ACM Trans. on Computer Systems, 2003, 21(3): 270–313. [doi: 10.1145/859716.859719]

[12] 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: ACM, 2018. 561–575. [doi: 10.1145/3230543.3230544]

[13] 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]

[14] 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]

[15] Dai M, Cheng G, Zhou YY. Survey on measurement methods in software-defined networking. Ruan Jian Xue Bao/Journal of Software, 2019, 30(6): 1853−1874 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5832.htm [doi: 10.13328/j.cnki.jos.005832]

[16] Tan LZ, Su W, Zhang W, Lv JH, Zhang ZY, Miao JY, Liu XX, Li N. In-band network telemetry: A survey. Computer Networks, 2021, 186: 107763. [doi: 10.1016/j.comnet.2020.107763]

[17] Mohan V, Reddy YRJ, Kalpana K. Active and passive network measurements: A survey. Int'l Journal of Computer Science and Information Technologies, 2011, 2(4): 1372–1385.

[18] Wang M, Li B, Li Z. sFlow: Towards resource-efficient and agile service federation in service overlay networks. In: Proc. of the 24th Int'l Conf. on Distributed Computing Systems. Tokyo: IEEE, 2004. 628–635. [doi: 10.1109/ICDCS.2004.1281630]

[19] CISCO. Introduction to Cisco IOS NetFlow—A technical overview. 2012. https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html

[20] Metwally A, Agrawal D, El Abbadi A. Efficient computation of frequent and Top-*k* elements in data streams. In: Proc. of the 10th Int'l Conf. on Database Theory. Edinburgh: Springer, 2005. 398–412. [doi: 10.1007/978-3-540-30570-5_27]

[21] Karp RM, Shenker S, Papadimitriou CH. A simple algorithm for finding frequent elements in streams and bags. ACM Trans. on Database Systems, 2003, 28(1): 51–55. [doi: 10.1145/762471.762473]

[22] Demaine ED, López-Ortiz A, Munro JI. Frequency estimation of internet packet streams with limited space. In: Proc. of the 10th European Symp. on Algorithms. Rome: Springer, 2002. 348–360. [doi: 10.1007/3-540-45749-6_33]

[23] Manku GS, Motwani R. Approximate frequency counts over data streams. In: Proc. of the 28th Int'l Conf. on Very Large Data Bases. Hong Kong: VLDB Endowment, 2002. 346–357.

[24] Ting D. Data sketches for disaggregated subset sum and frequent item estimation. In: Proc. of the 2018 Int'l Conf. on Management of Data. Houston: Association for Computing Machinery, 2018. 1129–1140. [doi: 10.1145/3183713.3183759]

[25] Charikar M, Chen K, Farach-Colton M. Finding frequent items in data streams. In: Proc. of the 29th Int'l Colloquium on Automata, Languages, and Programming. Malaga: Springer, 2002. 693–703. [doi: 10.1007/3-540-45465-9_59]

[26] 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]

[27] Yu ML, Jose L, Miao R. Software defined traffic measurement with OpenSketch. In: Proc. of the 10th USENIX Conf. on Networked Systems Design and Implementation. Lombard: USENIX Association, 2013. 29–42.

[28] 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. Association for Computing Machinery, 2020. 1574–1584. [doi: 10.1145/3394486.3403208]

[29] Li YL, Miao R, Kim C, Yu ML. FlowRadar: A better NetFlow for data centers. In: Proc. of the 13th USENIX Conf. on Networked Systems Design and Implementation. Santa Clara: USENIX Association, 2016. 311–324.

[30] Liu ZX, Manousis A, Vorsanger G, Sekar V, Braverman V. One sketch to rule them all: Rethinking network flow monitoring with UnivMon. In: Proc. of the 2016 ACM SIGCOMM Conf. Florianopolis: Association for Computing Machinery, 2016. 101–114. [doi: 10.1145/2934872.2934906]

[31] 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]

[32] 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]

[33] Huang Q, Sun HF, Lee PPC, Bai W, Zhu F, Bao YG. OmniMon: Re-architecting network telemetry with resource efficiency and full accuracy. In: Proc. of the 2020 Annual Conf. of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication. New York: Association for Computing Machinery, 2020. 404–421. [doi: 10.1145/3387514.3405877]

[34] Kučera J, Popescu DA, Wang H, Moore A, Kořenek J, Antichi G. Enabling event-triggered data plane monitoring. In: Proc. of the Symp. on SDN Research. San Jose: Association for Computing Machinery, 2020. 14–26. [doi: 10.1145/3373360.3380830]

[35] Sivaraman V, Narayana S, Rottenstreich O, Muthukrishnan S, Rexford J. Heavy-hitter detection entirely in the data plane. In: Proc. of the 2017 Symp. on SDN Research. Santa Clara: Association for Computing Machinery, 2017. 164–176. [doi: 10.1145/3050220.3063772]

[36] Zhang YD, Liu ZX, Wang RX, Yang T, Li JZ, Miao RJ, Liu P, Zhang RW, Jiang JC. CocoSketch: High-performance sketch-based measurement over arbitrary partial key query. In: Proc. of the 2021 ACM SIGCOMM Conf. Association for Computing Machinery, 2021. 207–222. [doi: 10.1145/3452296.3472892]

[37] Lee S, Levanti K, Kim HS. Network monitoring: Present and future. Computer Networks, 2014, 65: 84–98. [doi: 10.1016/j.comnet.2014.03.007]

[38] Leiner BM, Cerf VG, Clark DD, Kahn RE, Kleinrock L, Lynch DC, Postel J, Roberts LG, Wolff S. A brief history of the internet. ACM SIGCOMM Computer Communication Review, 2009, 39(5): 22–31. [doi: 10.1145/1629607.1629613]

[39] McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69–74. [doi: 10.1145/1355734.1355746]

[40] Doria A, Salim JH, Haas R, Khosravi H, Wang W, Dong L, Gopal R, Halpern J. Forwarding and Control Element Separation (ForCES) Protocol Specification. IETF RFC 5810, 2010. [doi: 10.17487/RFC5810]

[41] Pfaff B, Davie B. The Open vSwitch Database Management Protocol. IETF RFC 7047, 2013. [doi: 10.17487/RFC7047]

[42] Pfaff B, Pettit J, Koponen T, Jackson EJ, Zhou A, Rajahalme J, Gross J, Wang A, Stringer J, Shelar P, Amidon K, Casado M. The design and implementation of Open vSwitch. In. Proc. of the 12th USENIX Symp. on Networked Systems Design and Implementation. Oakland: USENIX Association, 2015. 117–130.

[43] Lockwood JW, McKeown N, Watson G, Gibb G, Hartke P, Naous J, Raghuraman R, Luo J. NetFPGA—An open platform for gigabit-rate network switching and routing. In: Proc. of the 2007 IEEE Int'l Conf. on Microelectronic Systems Education. San Diego: IEEE, 2007. 160–161. [doi: 10.1109/MSE.2007.69]

[44] Bosshart P, Daly D, Gibb G, Izzard M, McKeown N, Rexford J, Schlesinger C, Talayco D, Vahdat A, Varghese G, Walker D. P4: Programming protocol-independent packet processors. ACM SIGCOMM Computer Communication Review, 2014, 44(3): 87–95. [doi: 10.1145/2656877.2656890]

[45] Estan C, Keys K, Moore D, Varghese G. Building a better NetFlow. ACM SIGCOMM Computer Communication Review, 2004, 34(4): 245–256. [doi: 10.1145/1030194.1015495]

[46] Whang KY, Vander-Zanden BT, Taylor HM. A linear-time probabilistic counting algorithm for database applications. ACM Trans. on Database Systems, 1990, 15(2): 208–229. [doi: 10.1145/78922.78925]

[47] Durand M, Flajolet P. Loglog counting of large cardinalities. In: Proc. of the 11th European Symp. on Algorithms. Budapest: Springer, 2003. 605–617. [doi: 10.1007/978-3-540-39658-1_55]

[48] Heule S, Nunkesser M, Hall A. HyperLogLog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In: Proc. of the 16th Int'l Conf. on Extending Database Technology. Genoa: ACM, 2013. 683–692. [doi: 10.1145/2452376.2452456]

[49] Misra J, Gries D. Finding repeated elements. Science of Computer Programming, 1982, 2(2): 143–152. [doi: 10.1016/0167-6423(82)90012-0]

[50] Bloom BH. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970, 13(7): 422–426. [doi: 10.1145/362686.362692]

[51] Agarwal PK, Cormode G, Huang ZF, Phillips JM, Wei ZW, Yi K. Mergeable summaries. ACM Trans. on Database Systems, 2013, 38(4): 26. [doi: 10.1145/2500128]

[52] Mandal A, Jiang H, Shrivastava A, Sarkar V. Topkapi: Parallel and fast sketches for finding Top-*k* frequent elements. In: Proc. of the 32nd Int'l Conf. on Neural Information Processing Systems. Montréal: Curran Associates Inc., 2018. 10921–10931.

[53] Lu JY, Pan T, He S, Miao M, Zhou GZ, Qi YN, Lyu B, Zhu SM. A two-stage heavy hitter detection system based on CPU spikes at cloud-scale gateways. In: Proc. of the 41st Int'l Conf. on Distributed Computing Systems. Washington: IEEE, 2021. 348–358. [doi: 10.1109/ICDCS51616.2021.00041]

[54] Pan T, Yu NB, Jia CH, Pi JW, Xu L, Qiao YS, Li ZG, Liu K, Lu J, Lu JY, Song EG, Zhang J, Huang T, Zhu SM. Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches. In: Proc. of the 2021 ACM SIGCOMM Conf. ACM, 2021. 194–206.

[55] Suh J, Kwon TT, Dixon C, Felter W, Carter J. OpenSample: A low-latency, sampling-based measurement platform for commodity SDN. In: Proc. of the 34th IEEE Int'l Conf. on Distributed Computing Systems. Madrid: IEEE, 2014. 228–237. [doi: 10.1109/ICDCS.2014.31]

[56] Chen XQ, Landau-Feibish S, Braverman M, Rexford J. BeauCoup: Answering many network traffic queries, one memory update at a time. In: Proc. of the 2020 Annual Conf. of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication. ACM, 2020. 226–239.

[57] Harrison R, Cai QZ, Gupta A, Rexford J. Network-wide heavy hitter detection with commodity switches. In: Proc. of the 2018 Symp. on SDN Research. Los Angeles: ACM, 2018. 8. [doi: 10.1145/3185467.3185476]

[58] Huang Q, Sheng SY, Chen X, Bao YG, Zhang R, Xu YW, Zhang G. Toward nearly-zero-error sketching via compressive sensing. In: Proc. of the 18th USENIX Symp. on Networked Systems Design and Implementation. USENIX Association, 2021. 1027–1044.

[59] Høiland-Jørgensen T, Brouer JD, Borkmann D, Fastabend J, Herbert T, Ahern D, Miller D. The eXpress data path: Fast programmable packet processing in the operating system kernel. In: Proc. of the 14th Int'l Conf. on Emerging Networking Experiments and Technologies. Heraklion: ACM, 2018. 54–66. [doi: 10.1145/3281411.3281443]

[60] Gebara N, Lerner A, Yang MR, Yu ML, Costa P, Ghobadi M. Challenging the stateless quo of programmable switches. In: Proc. of the 19th ACM Workshop on Hot Topics in Networks. ACM, 2020. 153–159. [doi: 10.1145/3422604.3425928]

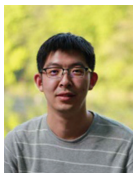
附中文参考文献:

[14] 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]

[15] 戴冕, 程光, 周余阳. 软件定义网络的测量方法研究. 软件学报, 2019, 30(6): 1853−1874. http://www.jos.org.cn/1000-9825/5832.htm [doi: 10.13328/j.cnki.jos.005832]

钱昊(1999－), 男, 硕士生, 主要研究领域为可编程网络.

陈贵海(1963－), 男, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为未来网络系统与协议, 无线网络结构与优化, 物联网与传感网, 新型计算机体系结构, 数据中心核心技术, 数据分析与处理.



郑嘉琦(1986－), 男, 博士, 助理教授, 博士生导师, CCF 高级会员, 主要研究领域为网络协议与优化.