中国科学: 信息科学 2023 年 第 53 卷 第 10 期: 1924–1944
SCIENTIA SINICA Informationis
论文

面向 SDN 网络的分布式轻量级大流检测算法

周京晶[1], 黄河[1*], 孙玉娥[2,3], 杜扬[1], 张博宇[1]

1. 苏州大学计算机科学与技术学院, 苏州 215006
2. 苏州大学轨道交通学院, 苏州 215131
3. 嵌入式系统与服务计算教育部重点实验室 (同济大学), 上海 201804
* 通信作者. E-mail: huangh@suda.edu.cn

收稿日期: 2022–10–07; 修回日期: 2022–12–19; 接受日期: 2023–01–31; 网络出版日期: 2023–10–13

国家自然科学基金 (批准号: 62332013, 62072322, 62202322, U20A20182)、同济大学嵌入式系统与服务计算教育部重点实验室开放课题 (批准号: ESSCKF 2022-05)、江苏省自然科学基金 (批准号: BK20210706) 和江苏省博士后科研资助 (批准号: 2021K165B) 项目

摘要 在 SDN (software-defined networks) 网络中检测大流对负载均衡、异常检测、流量工程等网络应用的实施及网络服务质量的提升至关重要. SDN 网络通常使用流表统计流量数据, 但由于存储流表的三态内容寻址寄存器 (ternary content addressable memory, TCAM) 资源有限, 仅凭流表无法从海量网络数据中识别出所有大流, 需要使用紧凑数据结构作为额外的测量模块来辅助检测. 现有的研究多考虑将测量模块部署在单台或边缘交换机上, 但交换机中的高速存储资源和计算资源极度紧缺, 处于高流量链路上的交换机会因执行测量任务承受过高的负载, 甚至影响交换机核心功能的执行. 为此, 本文提出了一种面向 SDN 网络的分布式轻量级大流检测方案. 该方案将网络流量测量任务分摊至全网交换机, 进而实现测量负载的均衡, 并结合所设计的轻量级测量模块, 有效降低了每台交换机执行测量任务所需的计算和存储开销. 实验结果证明该方案较已有最新研究成果具有更高的测量精度及更低的计算和存储开销.

关键词 SDN 网络, 网络流量测量, 大流检测, 分布式测量, Sketch

1 引言

软件定义网络 (software-defined networks, SDN) 是区别于传统网络的新型网络架构, 它能通过中心控制器为每条网络流提供细粒度的集中式控制 [1,2]. 我们通常将网络流定义为网络中具有相同流标签的数据包集合, 其中流标签可以按照实际需求决定, 一般为数据包首部字段的子集. 例如源地址 – 目的地址流是由相同源 IP 地址发送至相同目的 IP 地址的所有数据包抽象而成. 在 SDN 网络中测量

引用格式: 周京晶, 黄河, 孙玉娥, 等. 面向 SDN 网络的分布式轻量级大流检测算法. 中国科学: 信息科学, 2023, 53: 1924–1944, doi: 10.1360/SSI-2022-0387
Zhou J J, Huang H, Sun Y E, et al. A distributed and lightweight elephant flow detection algorithm for software-defined networks (in Chinese). Sci Sin Inform, 2023, 53: 1924–1944, doi: 10.1360/SSI-2022-0387

流的大小, 即测量流的数据包总数, 有利于 SDN 网络流级策略 (如 QoS 策略 [3]) 优势的发挥、网络管理应用的开展、网络性能的提升 [4]. 因此, 在 SDN 网络中进行流大小测量是一项重要的研究议题.

我们通常将网络中包含数据包个数较多的流称为大流, 少数大流往往占据了网络中绝大多数流量, 使得网络流量分布呈现偏斜性 (skewness) [5~8]. 以真实网络流量数据集 CAIDA19 [9] 为例, 将流按照流大小排序, 会发现排列在前 2% 的大流占据了网络总流量的 80%, 而排列在后 85% 的小流的流大小不超过 10. 显然, 大流对网络性能的影响较之小流更加显著, 如大流更容易引起网络拥塞. 因此, 在网络中检测大流是流大小测量里的一个基础且关键的任务, 它可以为许多网络管理应用 (如负载均衡、异常检测、流量工程) 提供决策依据 [3,10]. 大流检测通常包括 top-k 流检测以及过阈值流检测, 前者检测网络中数据包数目排列在前 k 位的流, 后者检测网络中流大小超过一定阈值的流, 它们都需要提供大流流标签以及流大小估计值.

然而, 虽然大流数目较少, 但要将其准确地从海量数据中识别出来仍有难度. 在 SDN 网络里, 储存在 TCAM (ternary content addressable memory) 中流表的计数器字段能准确记录匹配流的大小, 但 TCAM 价格高昂且非常耗电, 因此流表容量极其有限. 较高端的 Noviswitch 2150 交换机上仅有 16000 条支持 OpenFlow 协议的流表项 [11], 而网络中的流数目可超过 10^7 [9]. 因为我们无法在测量过程中预测流的大小, 所以仅凭流表无法准确地跟踪网络中的所有大流 [12,13]. 文献 [14] 考虑了这种流表受限的情况, 它会在边缘交换机上部署额外的测量模块, 使用能够匹配网络流速的片上高速存储资源 SRAM (static random access memory) 以追踪一个测量周期内的 top-k 流. 考虑到交换机中极度紧缺的计算和存储资源, 我们通常使用紧凑数据结构 (Sketch) 作为测量模块, 它能将所有流量信息压缩进固定大小的内存中, 并为每条流提供近似的流大小估计值 [6~8,10,15~27].

目前已有许多利用 Sketch 进行大流检测的研究. 最传统的 top-k 流检测使用基于计数器的 Sketch (如 CountMin [8], Count-Sketch [16]) 记录流大小信息, 利用最小堆追踪 top-k 流. 但每次更新 Sketch 都需要多次哈希 (Hash) 及内存访问操作, 频繁访问最小堆也使每个包的处理时间过长, 致使吞吐量过低. 后来一些研究直接将流标签存储在 Sketch 中, 它们将 Sketch 中的基本操作单元称为桶, 并通过一些精心设计的机制尽量确保桶内存储的是大流流标签和相应流大小 [7,18~22]. 这些研究一般通过实时查询流大小的方式来检测过阈值流, 并在测量周期结束后对流估计值排序得到 top-k 流. 然而, 它们直接将流标签保存在 Sketch 里, 当流标签定义的字段较多时会导致空间消耗过大. 为了解决这一问题, 一些研究会使用指纹技术将任意长度的流标签映射至 16 bit 的指纹编码 [7,19,21], 但对于紧缺的片上存储资源来说仍旧耗费过高, 且存在流标签哈希冲突的风险.

此外, 上述研究多只考虑测量通过单台交换机的流量, 无法获得全局的流量信息. 尽管通过在网络内多节点都部署这些 Sketch 的方式能够弥补其不足, 但各节点工作时相互独立, 会统计通过本身的所有流量, 容易导致对流的重复测量, 且这些研究也未提供多节点流量数据聚合方案. 文献 [10,28] 使用边缘交换机协同测量全网流量, 一条流中的每个数据包会在其通过的出入口交换机上各以 $\frac{1}{2}$ 的概率被记录, 这在一定程度上避免了重复测量. 然而, 网络流量的分布是不均衡的, 这会使测量任务在交换机中分配不均, 若不考虑或仅使用边缘交换机协同测量, 会导致处于流量较高的链路上的交换机因测量任务过重而承受较大的计算和存储开销, 甚至严重干扰交换机执行核心功能, 如更新流表、转发数据包等 [10]. 因此, 需要一个新的能够适应高速网络的测量模块及分布式大流检测方案, 该方案应当使用尽可能多的交换机来分担测量负载, 从而满足在低计算和存储开销下保持高检测精度的需求.

针对上述问题, 本文提出了一个面向 SDN 网络的分布式轻量级大流检测方案, 该方案利用全网交换机实施协同测量, 从而降低了每台交换机需要记录的流量, 使得因流量分布不均带来的高负载能被多台交换机分摊, 实现了网络流量测量负载的均衡. 此外, 该方案使用本文提出的轻量级测量模块. 相

比于已有的研究, 该模块空间效率更高. 对于每个数据包, 它最多更新一个计数器的值 (以往的研究大多需要为每个数据包操作多个计数器), 所需的计算开销更低, 处理速度更快. 对于流标签的记录, 我们既不使用最小堆, 也不将流标签记入测量模块, 而是采用交换机和控制器实时通讯的方式将大流流标签上传至控制器保存, 进而节省交换机存储流标签所需的空间. 本文主要贡献如下:

(1) 提出了新的分布式大流检测框架, 利用全网交换机协同测量网络流量, 弥补了用以流量测量的交换机过少而可能导致的交换机高计算和存储开销的缺陷. 该框架利用 SDN 网络的特性, 采用数据平面实时检测大流的方式将其流标签上传至控制平面, 最终由控制平面通过分布式估计方法确定大流.

(2) 设计了一种轻量级的测量模块. 该模块采用的映射机制进一步降低了模块实际需要记录的数据包数量, 并使模块每层的计数器数目和长度逐层递减, 实现了更高的空间资源利用率. 多数情况下, 该模块对接收到的每个数据包仅执行一次哈希操作, 其他情况下会执行两次哈希及两次内存访问操作 (读和写), 所需计算开销很小, 从而为交换机保留较多计算资源, 供其执行核心功能.

(3) 在简化的单链路网络拓扑以及 4 元 Fat-Tree [29] 网络拓扑下, 使用真实世界网络流量数据集进行大量实验评估, 实验结果显示本文提出的方案能够在较低计算开销及较小的内存空间下以较高精度准确识别网络中的大流, 且性能优于对比实验中的其他方法.

本文其他章节安排如下: 第 2 节介绍该领域相关研究工作. 第 3 节给出网络模型结构以及相关问题定义和设计目标. 第 4 节介绍本文提出的测量模块的结构和相应的操作. 第 5 节提出分布式大流检测框架、大流检测算法和分布式估计方法. 第 6 节对算法进行实验评估, 最后对全文进行总结和讨论.

2 相关工作

在网络流量测量领域, Sketch 处理速度快、内存消耗小的特点使其获得了广泛关注. 使用 Sketch 进行流量测量需要提供两种基本操作: 更新和查询, 前者将流信息存入 Sketch 中, 后者通过 Sketch 为流提供流量估计值. 目前主流的大流检测方法主要有以下 3 类:

• **记录所有流量, 从中筛选大流.** 这类方法一般使用基于计数器的 Sketch, 如 CountMin [8], CU-Sketch [15], Count-Sketch [16] 记录所有流的流量信息. 这些 Sketch 都包含多层计数器数组, 每层关联一个哈希函数, 用以将流哈希至特定计数器. 若使用它们进行大流检测, 需要使用最小堆保存 top-k 流, 并通过实时查询的方式保存过阈值流. 但 Sketch 中的哈希冲突可能将小流误判为大流, 每次对最小堆的更新也可能需要多次内存访问操作, 这会降低交换机处理数据包的速度.

• **仅记录部分流信息.** 不同于上述方法, 这类方法使用基于键值对 (key-value pair) 的 Sketch 来保存部分流信息, 其中键为流标签, 值为流计数. 较经典的 SpaceSaving [30] 使用一定长度的 stream-summary (SS) 结构保存流的标签及大小信息, 它将每一条未保存在 SS 中的流视为潜在的大流并将其记录下来. 当 SS 已满时, 它将其中计数最小的流替换为新流, 并将最小计数 +1 作为新流的大小. 最后, 它取 SS 中计数前 k 大的流作为 top-k 流. 显然, 当小流数目非常多时, SS 很容易被小流占满, 使得检测误差过高. 为此, 一些研究设计了特定机制使键值对尽量保存大流信息. 其中, HeavyKeeper [7] 提出的 “指数衰减” 机制能使桶内保存的计数较少的流以较大概率被衰减掉, 相反计数较大的流能保存更久. 这种策略能使大流以更高概率被保存在桶内. HeavyGuardian [19], DHS [21] 沿用了指数衰减机制. 前者采用了 heavy part 和 light part 的设计, heavy part 会利用多个 “护卫” 降低 “国王” (桶中具有最高计数的键值对) 被衰减的次数, 而 light part 则用于记录小流. 后者提出了动态数据结构, 能够动态地组织桶中的空间分布, 利用有限的资源存储更多的流信息. 然而, 指数衰减技术较难识别较晚出现的大流: 当 value 为 100 时, 衰减的概率低至 4×10^{-4}. WavingSketch [18] 每次随机地赋

给一条流 +1 或 −1 的符号位来选择对它所属桶中的 waving-counter 加或减, 并利用该计数器找出大流, 进而将大流信息保存在桶内若干个 cell 中. CountMax [10], MV-Sketch [22], Elastic-Sketch [20] 的 heavy part 分别设计了不同的票选机制来决定是否替换大流流标签, 但复杂的桶结构 [20,22] 会使 Sketch 空间占用较高. 其中, Elastic-Sketch 也采用了 light part 设计, 以达到同时保留小流信息的目的.

• **采用过滤思想, 分离大小流.** 为了降低数目众多的小流对大流的干扰, 文献 [6,31∼35] 采取过滤思想将大小流分离, 从而提高大流检测精度. 文献 [31,32] 分别通过流采样和 TCBF 过滤模块过滤大部分小流, 并将过滤后的流存入 LRU 队列, 出队时遵循最近最少访问原则. 但这两种方案在过滤时需要对每个包操作 1 ∼ 2 个布隆过滤器, 导致处理时间过长. ASketch [6], ColdFilter [33], LogLogFilter [34], LadderFilter [35] 通过设计过滤器将大流或小流从数据流中分离. 其中 ASketch 是在已有的 Sketch 之前设置一个队列存储大流, 流先被插入队列中, 若队列已满, 则与 Sketch 进行双向通信, 分离出大流并将其替换入队列. 但当大流数目较多时, 每次扫描队列的开销会过高. ColdFilter 设计了一个两层都为 CU-Sketch 的单向过滤器来过滤小流, 当过滤器报告某条流超过一定阈值后, 这条流才会被插入到特定 Sketch 中. 但 ColdFilter 采取的保守更新方案会产生多次哈希及多次内存访问操作, 计算及时间开销都过高. LogLogFilter 在 ColdFilter 的基础上提高了过滤范围, 但未完全解决 ColdFilter 的缺陷. LadderFilter 通过对数据流的研究发现一些不活跃的小流在一段时间后转变为大流的概率非常低, 于是基于此研究结果设计了多级 LRU 桶队列过滤器. 当桶满时, 优先出队最近最少使用的键值对, 达成对小流的过滤目的. 然而, 过滤器方案通常需要与已有的 Sketch 结合 (如 SpaceSaving), 相比直接进行大流检测的 Sketch, 会有更高的存储需求.

此外, 上述方法中除 CountMax 以外均未考虑协同或分布式测量. 正如引言所述, 若使用这些方法测量网络流量, 会产生大量重复计算, 且这些方法也未提供流量数据聚合方案. 虽然 CountMax 采取边缘交换机协同测量的策略, 但一条流最多也仅由两台交换机测量, 当流量较高时仍会导致交换机负载过大. 因此, 我们应考虑采取分布式协同测量的方式, 利用整个网络的交换机共同测量网络流量, 从而最大程度上缓解交换机的计算和存储压力, 解决部分链路流量过高带来的高测量负载的问题. 同时, 部署在交换机上的测量模块也应当满足高速网络下低计算和存储开销以及高精度检测的需求.

3 问题模型

3.1 网络模型

在 SDN 网络里, 为了实现细粒度的流控制, 流标签的定义通常包括源 IP 及目的 IP 地址二元组. 当某条流中的数据包到达交换机而又无法在流表中查找到匹配项时, 交换机会将数据包的包头信息上传至控制器, 控制器会根据具体需求在它的源端及目的端之间计算一条细粒度路径, 并下发包含具体流标签信息的流表项 (per-flow rule) 至这条路径上的所有交换机. 但网络中海量的流数目与有限的流表资源这一矛盾会使控制器频繁为无法匹配流表的流配置路径, 同时交换机也会频繁将新流表项替换入流表中, 这会大量消耗控制器和交换机的处理资源. 对此, 文献 [14] 提出了一种能使 SDN 网络具有高可扩展性的混合路由机制, 它将流表优先分配给少量大流进行路径部署, 让大量小流由传统路由表进行转发, 从而避免了控制器为流频繁规划路径的问题. 因为混合路由机制能够有效规避 SDN 网络中流表容量受限带来的一系列问题, 同时该机制也需要细粒度的流量信息计算路由方案, 所以选用采取混合路由机制的 SDN 网络作为网络模型, 并在此基础上设计大流检测算法.

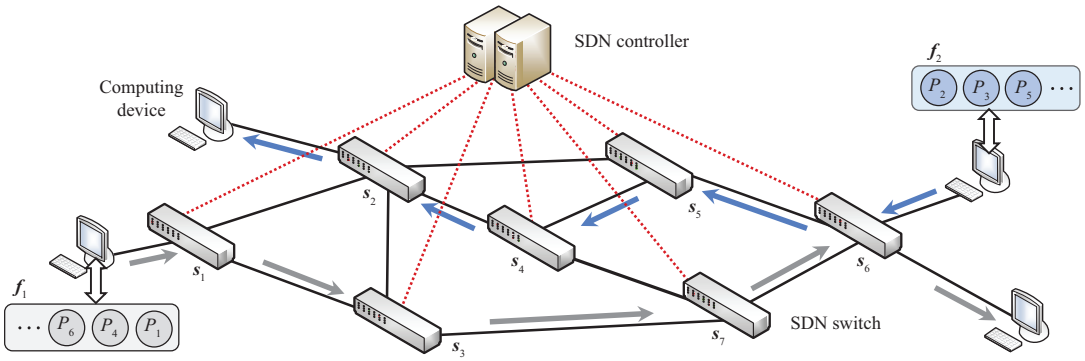

图 1　(网络版彩图) 网络模型, $S=\{s_1,s_2,\ldots,s_7\}$, $\mathcal{F}=\{f_1,f_2,\ldots\}$

Figure 1　(Color online) The network model, $S=\{s_1,s_2,\ldots,s_7\}$, $\mathcal{F}=\{f_1,f_2,\ldots\}$

如图 1 所示, 在网络模型中, 控制器和 SDN 交换机分别构成控制平面和数据平面. 我们将网络拓扑建模为图 $G=(S,E)$. 其中, $S=\{s_1,s_2,\ldots,s_m\}$ 是 SDN 交换机集合, E 表示交换机之间的通信链路. 将时间划分为多个测量周期, 每个测量周期内的流量由一组数据包 $\mathcal{P}=(\mathbb{P}_1,\mathbb{P}_2,\ldots,\mathbb{P}_N)$ 构成, 每个包 $\mathbb{P}_i$ 都携带流标签 f. 我们将流标签定义为 (源 IP 地址, 目的 IP 地址) 二元组. 具有相同流标签 f_j 的数据包集合构成流 f_j, 定义该集合的长度 n_{f_j} 为流 f_j 的大小, 其中 $f_j\in\mathcal{F}=\{f_1,f_2,\ldots,f_e\}$, $\mathcal{F}$ 为流集合.

根据文献 [14] 提出的混合路由机制, 每台交换机上有一张流表和传统路由表, 它们都由控制器下发, 其中路由表由控制器根据特定的路由协议计算得到. 对于每一个到达的数据包, 交换机会抽取它的流标签, 首先尝试流表匹配转发, 如若失败, 交换机将根据路由表转发数据包. 我们在每台交换机上部署流量测量模块, 由于流表会为其表内的流记录准确的流量值, 所以测量模块仅记录由路由表转发的流量. 在每个周期开始时, 测量模块内的计数器清零, 再开始本周期内的流量测量.

3.2　问题描述

在每个周期开始时, 控制器需要为每台交换机下发流表, 流表中的条目应该优先分配给大流, 但我们无法预测当前周期哪些流为大流, 故将参考上一周期流表以及测量模块的检测结果, 将它们检测出的 top-k 流作为本周期流表部署的依据. 同时, 由于网络流量千变万化, 当前周期中可能出现新的未被流表记录的大流, 若继续由路由表转发它们, 可能导致某些链路过载, 从而出现网络拥塞, 因此我们会检测当前周期内由路由转发的流大小超过一定阈值的流, 这一检测数据能提供给控制器现有的负载均衡模块或算法 (如 RDSR 算法 [14]) 使用, 该模块或算法通常会为大流重新规划较为空闲的链路. 综上, 本文同时考虑两种大流检测问题, 分别是为部署流表进行的单个周期内的全网 top-k 流检测, 其中 k 等于流表条目总数, 以及为负载均衡模块服务的过阈值流检测. 具体问题描述如下:

top-k 流检测. 在一个测量周期内, 将网络中的流按照流大小排序有 $(f_1,f_2,\ldots,f_e)$, 其中 $n_{f_1}\geqslant n_{f_2}\geqslant\cdots\geqslant n_{f_e}$. 给定整数 k, 需要检测出序列 $(f_1,f_2,\ldots,f_e)$ 中排列在前 k 位的流 $(f_1,f_2,\ldots,f_k)$.

过阈值流检测. 给定一个阈值 T, 如果一条流 f_i 的大小 $n_{f_i}>T$, 那么这条流是一条过阈值流. 过阈值流检测的目标是检测出一个周期内由路由表转发的所有过阈值流.

在采用混合路由机制的 SDN 网络中, 一个周期内的 top-k 流统计实际来源于流表计数与测量模块计数两部分. 流表记录的流量信息可以直接获得, 而本文的研究重点是如何设计一种好的测量方式及

测量模块来追踪由路由表转发的大流, 因此本文在后续内容中重点关注如何检测路由转发中前 k 大的流, 我们将这部分流定义为集合 Γ^k. 在分配流表时, 需要将 Γ^k 以及流表信息结合起来得到全网 top-k 流, 进而可以在新的周期开始时按照一定策略 (如贪心策略, 依次为排列在前的流分配余下最空闲的链路) 为它们制定 SDN 路径并下发相应流表项.

3.3 设计目标

由于现有的流量测量方法仅能利用少量交换机进行网络流量测量, 当部分链路流量较高时, 容易导致单台交换机因执行测量任务而承受过高的负载, 无法同时实现低计算和存储开销下的高精度检测. 因此, 我们的目标是设计一个分布式大流检测方案以及与之匹配的轻量级 Sketch, 从而将测量负载分摊给多台交换机, 进而实现降低单台交换机计算和存储开销的目的, 具体设计目标如下:

(1) 能够进行分布式检测, 尽量使用更多的交换机测量网络流量. 每条流将由多台交换机协作测量, 它包含的每个数据包应尽量只被某台交换机记录一次. 每台交换机仅记录每条流的部分流量, 每条流的流大小估计值应通过联合多台交换机记录的流信息及分布式估计方法得到.

(2) Sketch 的存储、计算消耗低, 处理单个包所需时间较短. 为了满足交换机片上高速存储资源紧缺的限制, Sketch 应当仅占用少量片上存储资源. 同时, 为了满足高速网络处理需求、匹配网络线速度, Sketch 处理单个包所需的哈希及内存访问次数应尽量降低, 从而节省计算资源并提高处理速度.

(3) 利用较少空间记录大流流标签. 网络中的流数目众多, 我们需要选择一种高效的方法从中识别出大流流标签并选择一种低存储开销的方式保存.

(4) 能够进行高精度检测. 测量精度的保证是实施各种应用的基础. 在本文研究的问题中, 应当尽量减少将小流错误识别为大流以及未识别出大流的情况出现的次数.

4 Sketch 结构及操作

本节首先介绍测量模块的设计思路, 然后描述模块结构细节以及相应的更新及查询操作.

4.1 Sketch 设计思路

当使用包含多层计数器数组的 Sketch (如 CountMin, Count-Sketch) 完成流量测量任务时, 每一层都会记录所有到达的流信息, 即一个数据包在每层数组中都被记录一次, 冗余量较大. 为了降低冗余, 我们参考流基数测量方法 PCSA (probabilistic counting with stochastic averaging) [36] 的映射机制, 将所有流信息以 $\frac{1}{2}$ 递减的比例存储在多层计数器数组中, 其中第 i 层存储流 $\frac{1}{2^{i+1}}$ 部分的信息 (层数从 0 开始). 具体来说, 将任意一条流的每个数据包以 $\frac{1}{2^{i+1}}$ 的概率映射至第 i 层, 那么第 i 层将预计记录 $\frac{N}{2^{i+1}}$ 个数据包. 采用这种方式, 一个数据包每次仅在某一层中被记录一次, 大大降低了计算开销.

从对每层映射概率的描述可以看出所有数据包都以较高概率被映射至低层, 因此小流里的绝大部分数据包被映射至低层数组中. 而大流由于数据包个数多, 即使在高层也有一定记录. 考虑到交换机资源有限, 我们不记录映射至低层的数据包, 即不为低层数组分配物理空间, 这一措施恰好在一定程度上起到了过滤小流的效果. 由于每层数组记录的数据包数目依次降低, 所以每层计数器个数也应递减, 为了进一步压缩空间, 每层计数器的位数也可随层数降低.

4.2 数据结构

根据上述思路, 本文提出了一个新的 Sketch, 它由 n 层计数器数组构成的 Array 以及 n 个对应层数的哈希函数 $\{h_0, h_1, \ldots, h_{n-1}\}$ 构成, Array 中每个元素都是一个计数器. 每层数组长度、计数器的

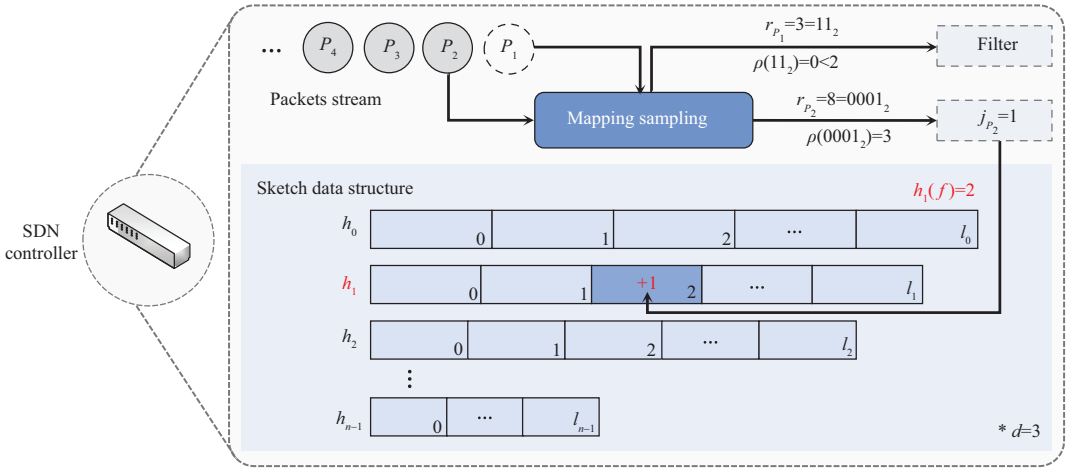

图 2　(网络版彩图) Sketch 结构及更新操作流程

Figure 2　(Color online) Data structure of Sketch and the update operation

位数依次递减. n 层数组的长度依次为 $l_0, l_1, \ldots, l_{n-1}$, 其中 $l_0 > l_1 > \cdots > l_{n-1}$. n 层计数器的位数依次为 $b_0, b_1, \ldots, b_{n-1}$, 其中 $b_0 \geqslant b_1 \geqslant \cdots \geqslant b_{n-1}$. 使用 h_i 为流找到其在第 i 层映射的计数器索引, 它能将任意流标签映射为一个 $[0, l_i - 1]$ 之间的随机整数. 此外, Sketch 还包含一个哈希函数 H, 它能将任意数据包映射为一个随机整数.

4.1 小节介绍每个数据包会以 $\frac{1}{2^{i+1}}$ 的概率被映射至第 i 层, 为了压缩空间, 不为低层数组分配物理存储空间. 假设丢弃逻辑第 0 至 $i-1$ 层, 从第 i 层开始记录, 那么逻辑第 i 层的真实物理层为第 0 层, 且每个数据包会以 $\frac{1}{2^{i+1}}$ 的概率映射至物理 0 层中. 我们仅考虑物理层中的映射关系, 使 $i+1=d$ (即丢弃 $0 \sim d-2$ 层逻辑层), 于是每个数据包会以 $\frac{1}{2^d}, \frac{1}{2^{d+1}}, \ldots, \frac{1}{2^{d+n-1}}$ 的概率映射至 Sketch 的第 $0, 1, \ldots, n-1$ 层中 (后续描述中如果层号不显式使用 "逻辑", 都默认为物理层号).

除 Sketch 外, 还需设置一个长度为 M 的位数组 B 以及哈希函数 H', $H'(\cdot) \in [0, M-1]$, 用以将任意流标签哈希至 B 中某一位 (具体用途将在 5.2 小节中介绍).

4.3　操作

Sketch 提供两种操作: 更新和查询, 下面将介绍两种操作的详细流程.

更新. 更新操作包含映射采样和记录两个步骤. 对于每一个到达的数据包 $\mathbb{P}$, 首先对其进行映射采样, 确保能以 $\frac{1}{2^{d+j}}$ 的概率映射至第 j 层. 映射采样成功后才能将它记录在 Sketch 中.

步骤 1: 映射采样. 对于每一个数据包 $\mathbb{P}$, 通过哈希函数 H 计算:

$$r = H(\mathbb{P}) \bmod 2^{n+d}. \tag{1}$$

计算 $i = \rho(r)$ 得到 $\mathbb{P}$ 被映射的逻辑层数 i, $\rho(r)$ 返回 r 二进制形式 $\langle r_0 r_1 \cdots r_{n+d-1} \rangle_2$ 左起首个 1 的位置 (如 $\rho(001_2) = 2$), 规定 $\rho(0) = n+d$. 这种方式能使每个数据包以 $(\frac{1}{2})^{\min(i+1, n+d)}$ 的概率映射至逻辑第 i 层. 自 4.2 小节可知逻辑第 $d-1$ 至 $d+n-2$ 层对应物理第 0 至 $n-1$ 层. 因此, 只有当 $d-1 \leqslant i \leqslant d+n-2$ 时, 数据包 $\mathbb{P}$ 映射采样成功, 并得到它被映射至测量模块中的物理层数 $j = i - d + 1$. 如图 2 所示, $d=3$, $\rho(r_{P_1}) = 0$ 导致 $\mathbb{P}_1$ 被过滤, 而 $\rho(r_{P_2}) = 3$ 使 $\mathbb{P}_2$ 映射采样成功.

步骤 2: 记录. 当数据包 $\mathbb{P}$ 映射采样成功后得到它本次被映射的层数 j, 接着根据 $\mathbb{P}$ 所属的流标签 f, 计算 $u_j = h_j(f)$ 得到它在第 j 层的计数器索引, 然后使 $\text{Array}[j][u_j] = \text{Array}[j][u_j] + 1$. 如图 2

所示, 数据包 $\mathbb{P}_2$ 被映射采样至第 1 层, 然后使第 $h_1(f)=2$ 个计数器递增 1. 由于数据包是以 $\frac{1}{2^{d+j}}$ 的概率被映射至第 j 层, 所以更新操作会返回该层给出的对流 f 的估计值 $v=$Array$[j][u_j]\times 2^{d+j}$ 用于检测初步判断. 更新算法细节见算法 1.

算法 1 更新

输入: $\mathbb{P}$: 数据包; f: $\mathbb{P}$ 所属流标签;

Function Update$(\mathbb{P}, f)$:
$r \Leftarrow H(\mathbb{P}) \bmod 2^{n+d}$;
$i \Leftarrow \rho(r)$;
$v \Leftarrow 0$; // 如果返回值 $v=0$ 意味着映射采样失败
if $d-1 \leqslant i \leqslant d+n-2$ **then**
  $j \Leftarrow i-d+1$;
  $u_j \Leftarrow h_j(f)$;
  Array$[j][u_j] \Leftarrow$ Array$[j][u_j]+1$;
  $v \Leftarrow$ Array$[j][u_j]\times 2^{d+j}$;
end if
Return v;

输出: v: 单层估计值.

查询. 对于任意流 f, Sketch 使用 n 层数组进行查询, 并返回当前 Sketch 给出的对流 f 的估计值. 对于每一层数组 j, 计算 $h_j(f)$, 使用 $\widehat{a_{f_j}}$ 作为流 f 在第 j 层的估计值, 有

$$\widehat{a_{f_j}} = \text{Array}[j][h_j(f)]\times 2^{d+j}. \tag{2}$$

取 n 层估计值的最小值 $\widehat{a_f}$ 作为 Sketch 对流 f 大小的估计值, 查询算法细节见算法 2:

$$\widehat{a_f} = \min_j\{\widehat{a_{f_j}}, 0\leqslant j\leqslant n-1\}. \tag{3}$$

算法 2 查询

输入: f: 流标签;

Function Query(f):
for $j \Leftarrow 0 \to n-1$ **do**
  $\widehat{a_{f_j}} \Leftarrow$ Array$[j][h_j(f)]\times 2^{d+j}$ // 计算第 j 层给出的估计值
end for
$\widehat{a_f} \Leftarrow \min\{\widehat{a_{f_j}}, \forall j\}$;
Return $\widehat{a_f}$;

输出: $\widehat{a_f}$: 对流 f 的多层估计值.

5 分布式检测算法

本节首先给出分布式检测框架, 然后介绍数据平面在该框架下利用第 4 节提出的 Sketch 执行的大流检测算法, 最后给出控制平面在该框架下对流的分布式估计方法.

5.1 分布式检测框架

本小节介绍如何利用 SDN 网络以及混合路由机制实现网络流量的分布式检测. 在分布式检测框架中, 每台交换机上都使用 SRAM 资源部署具有相同参数的 Sketch 以及相同长度的位数组 B, 但不

同交换机上使用不同的哈希函数 H, H' 以及 $\{h_0, h_1, \ldots, h_{n-1}\}$. 检测时, 由交换机构成的数据平面统计网络流量信息并捕获候选大流流标签, 控制平面负责确认检测结果.

在网络 G 中, 数据平面部署的 Sketch 仅记录由路由表转发的流, 而一张路由表中会记录到各个端网络的 "代价". 根据不同的路由协议, "代价" 具有不同含义, 但我们统一将 "代价" 定义为跳数以便执行分布式检测方案 (如定义 OSPF 协议中两节点之间的链路度量为 1, 或在路由表中添加额外的字段). 规定直连路由的跳数为 0, 如此每台交换机 s_i 上的路由表中会拥有这台交换机到每一条流 f 两端点 (源 IP、目的 IP) 所属网络的跳数 $P_{i,f,s}$, $P_{i,f,d}$, 那么就可以得出这条流在网络中转发的路径长度:

$$P_f = P_{i,f,s} + P_{i,f,d} + 1, \tag{4}$$

其中 1 表示交换机 s_i. 在一个较短的测量周期内, 可以认为路由部署没有产生变化, 因此由路由转发的流路径不变, 且其路径上的任意一台交换机都可计算它在网络中转发路径总长度 P_f. 我们考虑让流 f 的转发路径上所有交换机来协作测量它的流量, 其路径上每台交换机以 $\frac{1}{P_f}$ 的概率对流 f 的每个数据包进行采样, 如此流 f 每个数据包被 P_f 台交换机总采样次数的期望为 1, 极大降低了数据冗余. 只有数据包通过采样, 它才能进一步被 Sketch 执行更新操作.

我们需要利用数据平面检测出 top-k 流及过阈值流, 并提供流标签. 在 SDN 网络中, 控制器和交换机可通过南向接口的上下行通道通信, 因此采用交换机实时将检测到的大流流标签上传至控制器的方式来保存它们, 从而节省一部分本地存储资源. 而流估计值由控制平面联合这条流的路径上的交换机中记录的流信息以及分布式估计方法确定.

获取 top-k 流的重点在于获得 Γ^k. 为此我们需要在每个测量周期开始前设置一个 top-k 流阈值 T_k, 以检测路由转发中超过该阈值的流, 这部分流称为候选 top-k 流, Γ^k 将从中产生 (候选 top-k 流个数应高于 k). 而每台交换机针对每条流 f 的 top-k 阈值取决于这条流经过的路径长度 P_f, 流 f 路径上的交换机对流 f 检测的 top-k 阈值为 $T_{k,f} = \frac{T_k}{P_f}$. 如果某台交换机检测到流 f 大小超过 $T_{k,f}$, 则将流 f 的流标签上传至控制器保存. 为了区分候选 top-k 流以及过阈值流, 交换机在上传流标签时还会一同上传一位标志位, 我们以 0 表示候选 top-k 流, 1 表示过阈值流. 数据平面通过上述方法收集候选 top-k 流标签, 控制平面负责在测量周期结束后对每条候选 top-k 流进行分布式估计得到流估计大小, 最后排序取前 k 条得到 Γ^k, 再结合流表记录的流信息, 得到全网 top-k 流.

3.2 小节设定由路由转发的过阈值流阈值为 T. 同理, 在过阈值流检测中, 每台交换机针对每条流 f 的阈值为 $T_f = \frac{T}{P_f}$. 如果某台交换机检测到流 f 大小超过 T_f, 同样会将流 f 标签及值为 1 的标志位上传至控制器作为候选过阈值流. 由于检测过阈值流是为当前周期服务, 因此在控制器收到标志位为 1 的流标签时, 会进行实时分布式估计. 如果根据测量结果判断流 f 大小超过 T, 那么控制器将它识别为过阈值流. 我们默认负载均衡模块将为过阈值流分配流表项, 并将计数器设置为对它的分布式估计大小值, 之后这条流由流表直接计数, 不再通过 Sketch. 需要注意的是一条流在被检测为过阈值流之前可能已被识别为候选 top-k 流, 在这种情况下, 周期结束后这条流会有流表和 Sketch 给出的两种计数结果, 我们以流表计数为准.

5.2 数据平面大流检测算法

本小节将详细介绍数据平面的大流检测流程. 数据平面对 top-k 流和过阈值流的检测都为两阶段. 接下来将分别介绍实时 top-k 流检测和实时过阈值流检测流程.

实时 top-k 流检测. 在每台交换机上, 每当成功记录一个数据包 $\mathbb{P}$ (流标签为 f) 后, 都需要对流 f 进行一次检测. 第 1 阶段使用更新 $\mathbb{P}$ 后 Sketch 给出的对流 f 的估计值 v 判断, 如果 v 高于 $T_{k,f}$,

则进入第 2 阶段. 第 2 阶段检测会通过 Sketch 的查询操作获得流 f 的多层估计值 $\widehat{a_f}$, 如果 $\widehat{a_f} \geqslant T_{k,f}$, 则交换机判断 f 是一个候选 top-k 流, 将该流标签以及值为 0 的标志位上传至控制器.

为了避免对一条流重复多次的查询及上传带来的计算和通信资源的浪费, 每条流应只在首次被判断为候选 top-k 流时上传一次. 我们通过位数组 B 达成此目的. 在每个测量周期开始时初始化 B 为 0, 当某条流 f 通过一阶段检测后, 使用哈希函数 H' 计算 $h = H'(f)$, 通过 $B[h]$ 的值来判断流 f 是否已被上传: 若 $B[h] = 1$, 意味着流 f 已经上传, 或是因为哈希冲突导致别的流先将 $B[h]$ 置 1, 出现了假阳情况, 而无论是哪种原因, 我们都视流 f 已被上传, 不再做其他操作; 若 $B[h] = 0$ 则意味着 f 是首次出现, 继续进入二阶段检测, 若通过二阶段检测则上传 $(f, 0)$ 至控制器, 并置 $B[h] = 1$, 从而保证流 f 不再被重复执行二阶段查询操作及上传.

实时过阈值流检测. 实时过阈值流检测流程与实时 top-k 流检测流程类似. 在记录一个数据包 $\mathbb{P}$ 后, 首先进行一阶段检测, 判断 $\mathbb{P}$ 所属流 f 是否存在 $v \geqslant T_f$, 若成立, 则进入二阶段检测. 如果二阶段使用查询操作获得多层估计值 $\widehat{a_f}$ 后仍存在 $\widehat{a_f} \geqslant T_f$, 则上传 $(f, 1)$ 至控制器. 数据平面大流检测算法细节见算法 3.

算法 3 数据平面大流检测算法

输入: $\mathbb{P}$: 数据包; T_k: top-k 流阈值; T: 过阈值流阈值.

Function ElephantFlowDetection($\mathbb{P}, T_k, T$):
  提取数据包 $\mathbb{P}$ 的流标签 f;
  计算流 f 路由路径长度 P_f;
  产生一个随机整数 $r \in [0, 1)$ 进行采样;
  if $r < \frac{1}{P_f}$ **then**
    $v \Leftarrow$ Update($\mathbb{P}, f$); // 采样成功
    if $v \geqslant \frac{T_k}{P_f}$ **then**
      if $B[H'(f)] \neq 1$ **then**
        $\widehat{a_f} \Leftarrow$ Query(f); // 流 f 还未被上传
        if $\widehat{a_f} \geqslant \frac{T_k}{P_f}$ **then**
          上传 $(f, 0)$ 至控制器;
          $B[H'(f)] \Leftarrow 1$;
        end if
      end if
    end if
    if $v \geqslant \frac{T}{P_f}$ **then**
      $\widehat{a_f} \Leftarrow$ Query(f);
      if $\widehat{a_f} \geqslant \frac{T}{P_f}$ **then**
        上传 $(f, 1)$ 至控制器;
      end if
    end if
  end if

5.3 控制平面分布式估计方法

由于被交换机采样成功的数据包能够以 $\frac{1}{2^{d+j}}$ 的概率被映射至 Sketch 的第 j 层, 因此被 Sketch 记录的数据包与被交换机采样的数据包的比值 V 为

$$V = \sum_{j=0}^{n-1} \frac{1}{2^{d+j}} = \frac{1}{2^{d-1}} - \frac{1}{2^{d+n-1}}. \tag{5}$$

Sketch 对某条流 f 记录的总数为 $R_f=\sum_{j=0}^{n-1}\text{Array}[j][h_j(f)]$, 于是可以估计单台交换机对流 f 采样的总数为 R_f/V. 由于每个计数器可能由多条流共享, R_f 会偏大, 若直接累加多台交换机的采样数, 容易导致对流 f 的总估计值偏高. 为了降低误差, 采取如下分布式检测估计方法, 来确定候选大流的流大小估计值.

在 SDN 网络中, 控制器了解全网拓扑结构, 它清楚每条流具体通过哪几台交换机, 因此在对流 f 的分布式估计中, 控制器会请求其通过的 P_f 台交换机 $\{s_{f_1},s_{f_2},\ldots,s_{f_{P_f}}\}$ 上传其 Sketch 中每层记录了流 f 大小信息的计数器的值 $(c_{0,i},\ldots,c_{j,i},\ldots,c_{n-1,i})_f$, 其中 $c_{j,i}$ 表示交换机 s_{fi} 上的 Sketch 中有关流 f 的第 j 层计数器值 $\text{Array}[j][h_j(f)]$. 于是, 控制器获得一个关于流 f 的 $n\times P_f$ 计数器矩阵 C_f:

$$C_f=\begin{bmatrix} c_{0,1} & \cdots & c_{0,i} & \cdots & c_{0,P_f} \\ \vdots & \ddots & \vdots & & \vdots \\ c_{j,1} & \cdots & c_{j,i} & \cdots & c_{j,P_f} \\ \vdots & & \vdots & \ddots & \vdots \\ c_{n-1,1} & \cdots & c_{n-1,i} & \cdots & c_{n-1,P_f} \end{bmatrix}. \tag{6}$$

我们选取 C_f 中每行具有最小计数的计数器组成新的计数器组 $(c_0,c_1,\ldots,c_{n-1})_f$, 其中 c_j 为

$$c_j=\min_i\{c_{j,i},1\leqslant i\leqslant P_f\}, \tag{7}$$

于是对流 f 的分布式估计值 $\widehat{A_f}$ 为

$$\widehat{A_f}=P_f\cdot\sum_{j=0}^{n-1}c_j\Big/V. \tag{8}$$

控制平面会使用 $\widehat{A_f}$ 作为对流 f 的最终大小估计值. 在 top-k 流检测中, 当测量周期结束时, 交换机为每一条候选 top-k 流按照上述分布式检测估计方法计算 $\widehat{A_f}$, 然后按照估计结果排序取前 k 条流作为 Γ^k, 再结合流表信息得到本周期 top-k 流检测结果. 在过阈值流检测中, 当交换机收到标志位为 1 的流标签 f 时, 立即为 f 计算 $\widehat{A_f}$, 如果 $\widehat{A_f}\geqslant T$, 那么将其识别为一条过阈值流, 这条流的数据将被提供给控制器中负责负载均衡的模块或算法使用.

6 性能评估

本文使用 2019 年 CAIDA 在 Equinix NYC 监视器上监听的匿名流量 [9] 数据作为数据集, 选用 4 个相关算法: CU-Sketch 及最小堆组合 (简称 CU+min-heap), HeavyKeeper, WavingSketch, CountMax 作为对比. 由于 CU+min-heap, HeavyKeeper, WavingSketch 属于非分布式测量方案, 仅能测量通过单台交换机的流量, 并未给出网络结构下的工作方案, 故我们使用一个简化的单链路网络拓扑来兼顾非分布式与 (边缘) 分布式测量方案, 从而使它们能够进行公平对比. 在简化拓扑中, 使用一分钟 CAIDA 数据集, 它包括 36144349 个数据包, 1821462 条流. 为了进一步验证分布式算法在分布式场景下的性能, 我们在 4 元 Fat-Tree [29] 网络拓扑下检测了边缘分布式方法 CountMax 和本文分布式方法的性能. Fat-Tree 拓扑实验使用 5 分钟 CAIDA 数据集, 它包括 183888816 个数据包, 7173139 条流. 本节首先给出评估算法性能的指标和相关实验参数设置, 再给出对比实验结果及相应的详细分析.

6.1 性能指标

我们使用吞吐量 (throughput)、平均相对误差 (average relative error, ARE), 以及文献 [37] 提供的两个指标: 假阴性率 (false negative rate, FNR) 和假阳性率 (false positive rate, FPR) 作为实验性能指标. 假阴和假阳的判定如下: 给定两个正整数 l 和 h, l 和 h 分别为缓冲下界和缓冲上界. 算法的目标为检测超过阈值 T 的大流 ($l < T < h$), 定义 n_f 为流 f 的真实大小, 若没有检测出 $n_f \geqslant h$ 的流 f, 则将 f 视为一例 false negative (FN); 若错误检测出 $n_f \leqslant l$ 的流 f , 则将 f 视为一例 false positive (FP). 我们使用 FNR 和 FPR 来评估各算法的检测精度.

为了进一步对比各方法的计算开销, 使用 "最大负载比率" 作为验证指标. 具体计算方式为: 在网络中具有最高负载, 即通过的数据包数目最高的交换机上分别执行各检测算法, 统计每个算法中需要被执行更新操作的数据包总数, 计算该总数与我们分布式检测算法要更新的数据包总数的比值作为各算法的最大负载比率. 显然, 我们的分布式算法的最大负载比率为 1. 通过最大负载比率可以直观地对比各方法的计算开销, 当算法的最大负载比率高于 1 时, 表明算法需要处理的数据包总数高于分布式算法, 那么它会有更多的计算需求, 导致计算开销更大. 各性能指标详细计算方式如下:

(1) 吞吐量. Throughput $= \frac{N}{t}$, 其中 N 表示待处理的数据包总数, t 表示算法处理 N 个数据包需要花费的总时间. 吞吐量单位为 Mpps (million packets per second). 吞吐量代表算法对数据包的处理速度, 吞吐量越高表明算法处理数据包的速度越快.

(2) 平均相对误差. ARE $= \frac{1}{|\Phi|}\sum_{f_i\in\Phi}\frac{|\widehat{n_{f_i}}-n_{f_i}|}{n_{f_i}}$, 其中 Φ 是算法检测出的 top-k 流集合 (包含过阈值流), $\widehat{n_{f_i}}$ 为流 f_i 的估计值; n_{f_i} 为流 f_i 的真实值. ARE 评估算法对流大小估计的平均误差比率.

(3) FNR. FNR $= \frac{|\mathrm{FN}|}{|H|}$, 其中 $|\mathrm{FN}|$ 为 FN 的个数, $|H|$ 为 $n_f \geqslant h$ 的流 f 的个数.

(4) FPR. FPR $= \frac{|\mathrm{FP}|}{|L|}$, 其中 $|\mathrm{FP}|$ 为 FP 的个数, $|L|$ 为 $n_f \leqslant l$ 的流 f 的个数.

(5) 最大负载比率. 在具有最高负载的交换机上, 若算法需要对数据包执行更新操作的总数为 x, 我们分布式算法需要对数据包执行更新操作的总数为 y, 那么该算法的最大负载比率为: $\frac{x}{y}$.

6.2 参数设置

本文使用两种网络拓扑来验证分布式检测算法的性能, 它们分别是简化的单链路网络拓扑以及 4 元 Fat-Tree [29] 网络拓扑. 在两种网络拓扑中, 假设所有流量都由路由表转发, 并利用不同算法进行 top-k 流以及过阈值流检测, 根据实验结果对比它们的性能. 两种拓扑的具体细节如下:

简化的单链路网络拓扑. 分布式测量的本质是使用多台交换机协作测量网络流量, 因此可以设置简化的单链路拓扑兼顾分布式方案与非分布式方案. 简化的单链路拓扑是将网络简化为一条链路, 这条链路上包含 m 台交换机, 每台都通过所有流量. 我们将链路两端的交换机设为边缘交换机. 非分布式方案 (CU+min-heap, HeavyKeeper, WavingSketch) 只需要利用链路中的一台交换机进行检测; 边缘分布式方案 (CountMax) 利用链路两端的边缘交换机进行检测; 本文的分布式测量方案利用链路上的 m 台交换机进行检测. 在简化拓扑中, 两类检测方案都能测量到所有流量, 保证了测量结果的公平性.

4 元 Fat-Tree [29] 网络拓扑. 考虑到非分布式方案无法在复杂网络拓扑下工作, 因此在 Fat-Tree 拓扑下仅使用 CountMax 作为对比. 我们将 p 元 Fat-Tree 拓扑的参数 p 设为 4, 那么该网络中包含 4 台核心交换机、8 台汇聚交换机、8 台边缘交换机, 其中汇聚交换机与边缘交换机共组成 4 个组, 每组包含两台汇聚及边缘交换机. 为了将 CAIDA 数据集中的真实流量数据整合到该拓扑下, 我们按照下列方法为数据集中的每条流选择一条路径作为路由选择结果, 以进行仿真实验.

表 1 d 与每层计数器的长度关系

Table 1 d and the length of counter

d	Layer index			
	0 (bit)	1 (bit)	2 (bit)	3 (bit)
4	12	12	12	12
5	12	12	12	8
6	12	12	8	8

Fat-Tree 拓扑下的路径选择. 对于每条流, 首先为它在 8 台边缘交换机中随机选择一台出口交换机和一台入口交换机, 根据出入口交换机的选择会有以下 3 种情况:

(1) 出入口交换机为同一台, 那么这条流仅需要这一台交换机转发, 路径长度为 1;

(2) 出入口交换机并非同一台, 但位于同一个组, 那么出入口交换机之间有 2 条需要经过同组汇聚交换机的路径 (p 元 Fat-Tree 拓扑中, 同组非同台边缘交换机之间的路径数目为 $\frac{p}{2}$) , 我们随机选择其中一条作为流的转发路径, 路径长度为 3;

(3) 出入口交换机位于不同组, 那么它们之间有 4 条通过汇聚及核心交换机的路径 (不同组的边缘交换机之间的路径数目为 $\frac{p^2}{4}$), 我们同样随机选择其中一条作为流的转发路径, 路径长度为 5.

当流的路径确定后, CountMax 只会利用这条路径上的一台或两台边缘交换机进行流量测量, 而本文的分布式方案将利用路径上的所有交换机来执行测量任务.

我们将在两种网络拓扑的同等实验条件下进行 top-k 流和实时过阈值流检测, 以评估本文提出的分布式检测算法的性能. 在计算 top-k 流的 FNR 和 FPR 时, 设定阈值 T' 为真实 top-k 流中的第 k 条流的大小, 缓冲下界 $l' = (1-\delta)\cdot T'$, 缓冲上界 $h' = (1+\delta)\cdot T'$; 对于过阈值流, 以 l 为变量, 设定 $h = (1+\delta)\cdot l$, $T = \frac{h+l}{2}$, 其中 $\delta = 0.2$.

根据文献 [38] 中的推荐, 将 CU-Sketch 的行数设定为 3, 计数器长度设定为 32 bit. 我们选用源 IP 及目的 IP 地址作为流标签, 因此最小堆中键为 64 bit, 值为计数器 32 bit. 最小堆的大小由 k 决定, 分配最小堆后剩下的内存将分配给 CU-Sketch, 决定其每层的计数器数目. HeavyKeeper, WavingSketch, CountMax 的相关参数均遵循其论文中的设置. 由于本文提出的 Sketch 为多层结构且每层计数器数目和计数器长度都不相同, 所以当空间给定时, Sketch 有多种分配方式, 其首先依赖于首层映射概率 $\frac{1}{2^d}$ 中的参数 d 及每层计数器的长度, 设定 d 与计数器长度关系如表 1 所示. 以表 1 第 1 行为例, 当 $d = 4$ 时, 第 0, 1, 2, 3 层计数器长度分别为 12, 12, 12, 12 bit. 实验中, 将 Sketch 的层数设为 3, 每种内存下各参数详细设置如表 2 所示. 每台交换机上位数组 B 的长度 M 设置为为 1 kB.

在简化网络拓扑中, 我们测试了在内存为 40 kB, $k = 500$, $l = 25000$ 的设置下参数 m 对分布式检测算法精度的影响. 在其他对比实验中, 设置 $m = 6$. 对于算法性能的评估, 测试了各算法在内存为 15 kB 条件下参数 k 不同时及 $k = 500$ 条件下内存不同时的吞吐量. 在精度评估中, 当内存不同时, 我们设置 $k = 500$, $l = 25000$, 测试了在内存空间为 15 ~ 40 kB 时各算法 top-k 流检测的 FPR, FNR, ARE 以及过阈值流检测的 FPR, FNR; 在 k 不同时, 设置内存为 40 kB, 测试了 k 为 250 ~ 1500 时各算法 top-k 流检测的 FPR, FNR, ARE; 在 l 不同时, 设置内存为 40 kB, $k = 500$, 测试了 l 为 15000 ~ 25000 情况下各算法过阈值流检测的 FPR, FNR; 在计算最大负载比率时, 设置内存为 15 kB.

在 Fat-Tree 网络拓扑中, 测试了在 $k = 500$, $l = 50000$ 的条件下, 内存为 15 ~ 40 kB 时 CountMax 与本文分布式算法 top-k 流检测的 FPR, FNR, ARE 以及过阈值流检测的 FPR, FNR. 此外, 在内存为 15 kB 时, 测试了 k 为 250 ~ 1500 时两个算法 top-k 流检测的 FPR, FNR, ARE, l 为 30000 ~ 50000

表 2 Sketch 参数设置

Table 2 Our Sketch's experimental parameter settings

Memory (kB)	Layer index			d
	0	1	2	
15	4200	3200	2200	5
20	5350	4350	3350	5
25	6450	5450	4450	5
30	7600	6600	5600	4
35	8750	7750	6750	4
40	10050	8850	7750	4

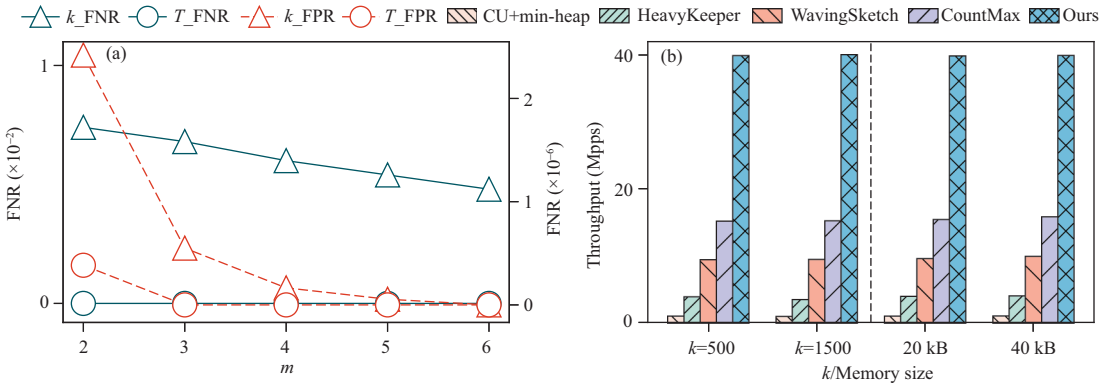

图 3 (网络版彩图) (a) 分布式检测精度; (b) 各算法吞吐量对比

Figure 3 (Color online) (a) Accuracy of distributed detection; (b) throughput comparison

时两个算法过阈值流检测的 FPR, FNR 以及最大负载比率. 所有实验结果为 10 次重复实验结果的平均值.

6.3 实验结果及分析

6.3.1 简化单链路网络拓扑

(1) 分布式检测性能. 图 3(a) 对比了使用不同数目的 m 台交换机进行分布式检测 top-k 流以及过阈值流 (图例中 k 和 T 分别表示 top-k 流及过阈值流) 的 FNR 以及 FPR. 图 3 表明本算法对 top-k 流以及过阈值流检测的假阴率和假阳率会随着参数 m 的增多而降低. 在 top-k 流检测中, 分布式检测的 FNR 最高为 0.0074, 最低为 0.0048, 而 FPR 最高为 2.42×10^{-6}, 最低为 0, 据统计高于 h' 的流有 353 条, 低于 l' 的流有 1820746 条 (h'=10641, l'=7094), 这意味着最高只有 2.6 (4.4) 条流被漏报 (多报), 最低仅 1.7 (0) 条流被漏报 (多报). 而在过阈值流检测中, 当 $m = 2$ 时, FPR 为 3.84×10^{-7}, 即在低于 l 的 1821379 条流中漏报 0.7 条. 其他条件下 FPR 以及 FNR 都为 0. 这意味着分布式检测过阈值流几乎没有出现漏报和多报.

(2) 吞吐量 (throughput). 使用搭载了 Intel Xeon E5-2643 v4@3.40GHz 处理器的服务器进行不同参数 k 及不同内存空间下的吞吐量实验. 从图 3(b) 可以看出当 k 或内存不同时, 各方法的吞吐量性能几乎不受上述两项参数的影响, 这是由于参数 k 的选择和空间大小的设置对数据包处理过程的影响极小. 同时, 在相同的参数设置下, 本文提出的分布式检测算法的吞吐量总是明显高于其他

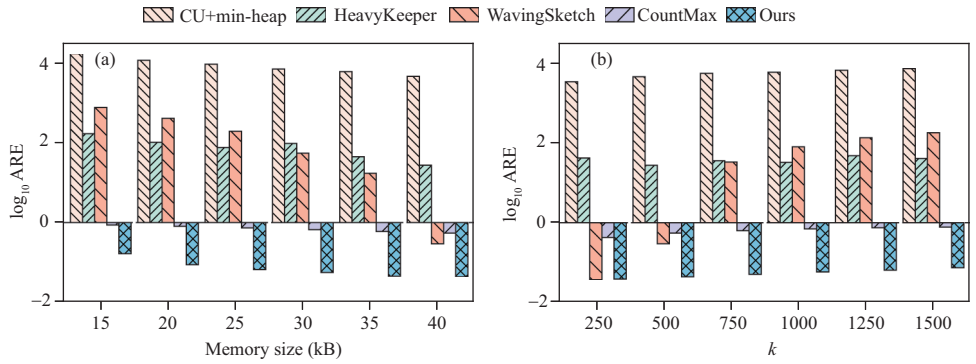

图 4　(网络版彩图) 不同条件下各算法 ARE

Figure 4　(Color online) ARE of algorithms under different memory or k. (a) ARE vs. memory; (b) ARE vs. k

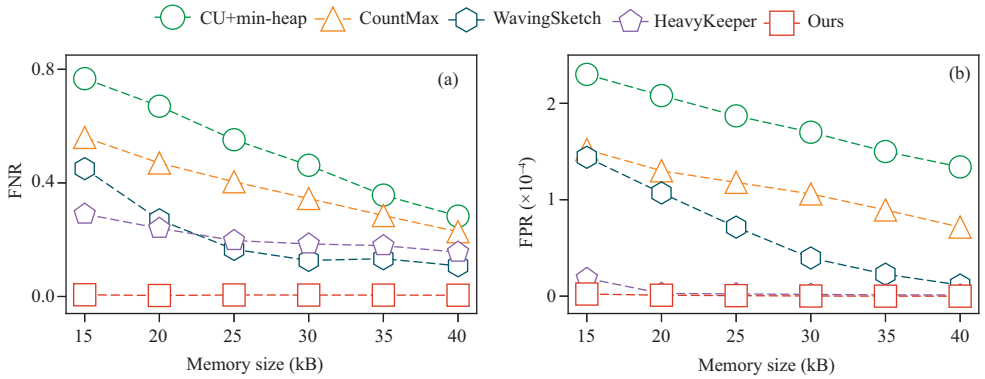

图 5　(网络版彩图) 内存不同时各算法 top-k 检测精度

Figure 5　(Color online) Accuracy of top-k detection under different memory. (a) FNR vs. memory; (b) FPR vs. memory

4 种算法, 它的吞吐量较 CountMax, WavingSketch, HeavyKeeper, CU+min-heap 分别高出 2.6, 4.2, 10, 38.5 倍. 分布式方案吞吐量最高是因为该方案能让每台交换机仅处理通过它的部分流量, 且方案中测量模块采用的映射机制能进一步加快对数据包的处理速度. CountMax 采取边缘交换机协同测量, 吞吐量次高. 而 HeavyKeeper 以及 CU+min-heap 因为涉及了堆操作, 吞吐量较低.

(3) 平均相对误差 (ARE). 图 4(a) 和 (b) 分别展示了在不同内存空间和不同参数 k 下 5 种算法在 top-k 流检测上的平均相对误差 (top-k 流包含了过阈值流), 发现分布式检测算法的 ARE 几乎始终处在最低水平. 当内存不同时, 本算法的 ARE 相较于 CU+min-heap, HeavyKeeper, WavingSketch, CountMax 低 96059 ~ 130192, 602 ~ 1623, 7 ~ 4358, 5 ~ 12 倍. 当 k 不同时, 仅在 $k = 250$ 时, 本算法的 ARE 与 WavingSketch 相差不大, 当 $k > 250$ 时, 本算法的 ARE 较 WavingSketch 低 7 ~ 2335 倍. 而本算法与其他 3 种算法的 ARE 差距与当内存不同时的实验结果相似.

(4) 不同内存下 top-k 流检测精度. 图 5(a) 和 (b) 对比了在不同内存空间下 5 种算法的 top-k 流检测精度. 相较于其他 4 种算法, 分布式检测算法拥有最低的 FNR 与 FPR. 在高 (低) 于 h' (l') 的 353 (1820746) 流中, 本算法最低仅漏报 (多报) 1.6 (0) 条流, 最高漏报 (多报) 2.3 (4) 条流, 这是因为分布式算法利用了多台交换机, 能提供更多的候选流标签, 同时分布式估计方法又保持了精度. 而 CU+min-heap, HeavyKeeper, WavingSketch, CountMax 最高会漏报 (多报) 271 (418), 102 (34), 160 (263), 198 (277) 条, 最低也会漏报 (多报) 100 (245), 55 (2), 38 (21), 81 (130) 条流.

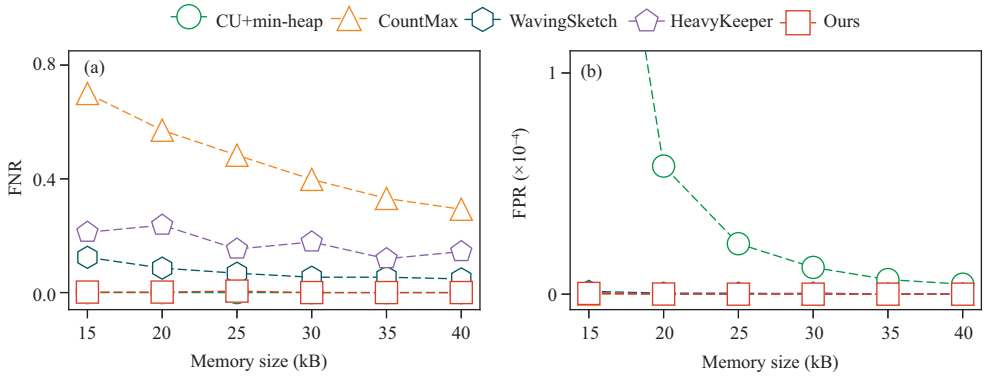

图 6 (网络版彩图) 内存不同时各算法过阈值流检测精度

Figure 6 (Color online) Accuracy of heavy hitter under different memory. (a) FNR vs. memory; (b) FPR vs. memory

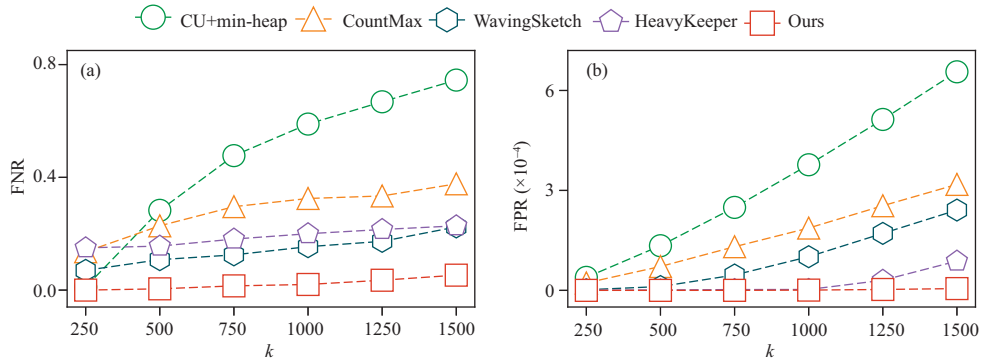

图 7 (网络版彩图) k 不同时各算法 top-k 检测精度

Figure 7 (Color online) Accuracy of top-k detection under different k. (a) FNR vs. k; (b) FPR vs. k

(5) 不同内存下过阈值流检测精度. 图 6(a) 和 (b) 对比了 5 种算法的过阈值流检测精度. HeavyKeeper 和 CountMax 对流的估计结果偏低, 因此它们不会出现假阳的情况, 故这两种算法的 FPR 为 0. 但相反的, 它们容易漏判而使 FNR 较高. CU-Sketch 对流的估计结果偏高, 它不会漏掉任何一条过阈值流, 因此它的 FNR 为 0, 但它非常容易将小流误判为一条过阈值流, 导致 FPR 非常高. 因此这 3 种算法在过阈值流的检测上并不稳定. 相比于 WavingSketch, 本算法在低于 l 的 1821379 条流中最多仅多报 0.6 条流, 而 WavingSketch 会多报 2.3 条流. 在高于 h 的 59 条流中最多会漏报 0.3 条流, 而 WavingSketch 最低 (高) 会漏报 2.8 (8) 条流. 在大部分情况下分布式检测算法不会产生漏报和多报, 这意味着本算法的过阈值流检测精度非常高.

(6) 参数 k 不同时 top-k 流检测精度. 图 7(a) 和 (b) 对比了当参数 k 不同时 5 种算法的 top-k 流检测精度, 当 k 增加时, 它们的 FNR 和 FPR 都有不同程度的上升, 但分布式算法的精度仍远高于其他算法. 当 $k = 250$ 时, 本算法不存在漏报和多报, 而 CU+min-heap, HeavyKeeper, WavingSketch, CountMax 则会漏报 (多报) 2 (71), 26 (1.2), 12 (4), 24 (41) 条流. 当 $k = 1500$ 时, 本算法仅漏报 (多报) 62.8 (9.7) 条流, 且多数 FN 来自于位数组哈希冲突. 而其他 4 种算法则会漏报 (多报) 892 (1193), 273 (160), 268 (439), 452 (579) 条流.

(7) 参数 l 不同时过阈值流检测精度. 图 8(a) 和 (b) 对比了缓冲下限 l 不同时 5 种算法的过阈值流检测精度. HeavyKeeper, CountMax, CU-Sketch 同样出现了 (4) 中描述的问题. 与 WavingSketch 相

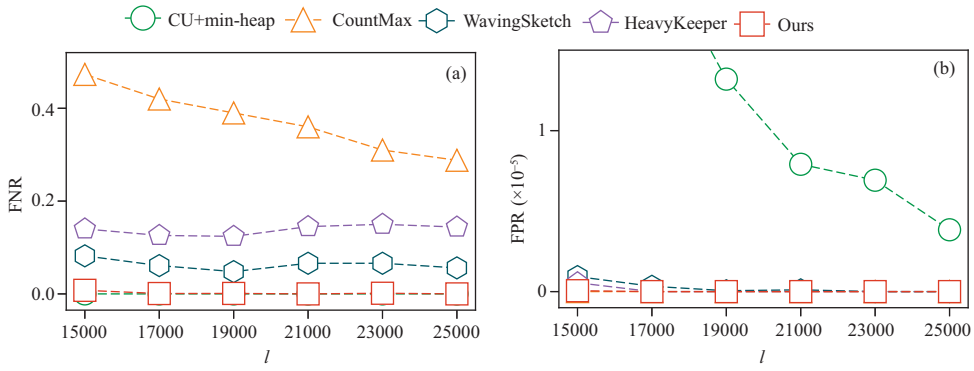

图 8　(网络版彩图) l 不同时各算法过阈值流检测精度

Figure 8　(Color online) Accuracy of heavy hitter under different l. (a) FNR vs. l; (b) FPR vs. l

表 3　两种网络拓扑下的最大负载比率

Table 3　Maximum load ratio under two network topologies

Algorithm	Simplified topology	Fat-tree topology
Ours	1	1
CM	3	1.9
WavingSketch	5.9	–
HeavyKeeper	5.9	–
CU+min-heap	5.9	–

比, 分布式算法仅在 $l = 15000$ 时多报 0.1 条流, 而前者会多报 1.7 条流. 同时分布式算法的 FNR 介于 0 ∼ 0.0077 之间 (漏报 0 ∼ 1 条流), 而 WavingSketch 的 FNR 在 0.048 ∼ 0.082 之间波动 (漏报 5 ∼ 12 条流). 因此, 本文提出的分布式检测算法拥有这 5 种算法中最稳定且最高的过阈值流检测精度.

(8) 简化拓扑下的最大负载比率. 表 3 列出了各算法在简化拓扑下的最大负载比率. 其他算法的最大负载比率远高于 1, 这是因为分布式算法会利用多台交换机分摊测量负载, 因此在分布式算法中需要被执行更新操作的数据包数量会远低于通过交换机的所有数据包数量, 而非分布式方法会对通过的所有数据包执行更新操作, 对计算需求最高.

6.3.2　4 元 Fat-Tree 网络拓扑

(1) 不同内存下大流检测精度. Fat-Tree 拓扑下分布式算法与 CountMax (CM) 的大流检测精度如表 4 所示, 其中表中每列左边的数据为 top-k 流检测精度, 右边为过阈值流检测精度. 显然, 分布式方案 top-k 流及过阈值流检测的 FNR 要远低于 CountMax, 分布式方案最高仅漏报 0.2 条流, 而 CountMax 最高会漏报 24 条流. 从 FPR 方面来看, 分布式方案仅在内存为 15 kB 时的过阈值流检测上漏报了 0.4 条流, 其他条件下均未出现假阳情况, 而 CountMax 在同等条件下的 top-k 流检测中会多报 15 条流. 此外, 分布式算法的 ARE 始终比 CountMax 低 6 倍左右.

(2) 参数 k 不同时 top-k 流检测精度. 当 k 不同时两种算法的 top-k 流检测精度如表 5 所示. 与简化拓扑中的实验结果类似, 当 k 增加时, 两种算法的 FNR, FPR, ARE 都有所上升, 但 CountMax 的各项指标始终都远差于分布式方案, 其中 FNR 及 FPR 较分布式方案高两个数量级.

(3) 参数 l 不同时过阈值流检测精度. 当 l 不同时两种算法的过阈值流检测精度如表 6 所示.

表 4 不同内存时大流检测精度, 左: top-k, 右: 过阈值

Table 4 Accuracy of elephant flow detection under different memory, left: top-k, right: heavy hitter

Memory (kB)	FNR (topk/T)		FPR (topk/T)		ARE	
	Ours	CM	Ours	CM	Ours	CM
15	5.5E−4/0	0.06/0.30	0/5.6E−8	2E−6/0	0.04	0.25
20	5.5E−4/0	0.04/0.22	0/0	4E−7/0	0.03	0.18
25	2.7E−4/0	0.03/0.12	0/0	0/0	0.02	0.14
30	5.5E−4/0	0.03/0.09	0/0	0/0	0.02	0.11
35	2.7E−4/0	0.02/0.06	0/0	0/0	0.02	0.09
40	0/0	0.01/0.04	0/0	0/0	0.02	0.08

表 5 k 不同时各算法 top-k 检测精度

Table 5 Accuracy of top-k detection under different k

k	FNR		FPR		ARE	
	Ours	CM	Ours	CM	Ours	CM
250	0	0.04	0	2.5E−7	0.03	0.18
500	5E−4	0.06	0	2.1E−6	0.04	0.24
750	1E−3	0.09	3E−8	5E−6	0.05	0.29
1000	1E−3	0.10	5E−8	1E−5	0.06	0.33
1250	2E−3	0.12	5E−8	1.7E−5	0.07	0.37
1500	4E−3	0.14	4E−7	2.3E−5	0.08	0.40

表 6 l 不同时各算法过阈值流检测精度

Table 6 Accuracy of heavy hitter under different l

l	FNR		FPR	
	Ours	CM	Ours	CM
30000	3.4E−4	0.39	6E−7	0
35000	0	0.36	4E−7	0
40000	0	0.35	7E−8	0
45000	0	0.3	7E−8	0
50000	0	0.3	6E−8	0

CountMax 会低估, 因此不会产生假阳. 虽然分布式算法会产生假阳, 但 FPR 始终非常低. 同时, 分布式算法几乎不产生假阴, 其最多仅漏报 0.2 条流, 而 CountMax 的 FNR 始终处在较高水平.

(4) Fat-Tree 拓扑下的最大负载比率. Fat-Tree 拓扑下的最大负载比率如表 3 所示. 因为非分布式方案无法在 Fat-Tree 拓扑下工作, 所以表中未列出非分布式算法在 Fat-Tree 拓扑下的最大负载比率. 从表 3 可以看出在 4 元 Fat-Tree 拓扑中, CountMax 的计算负载较分布式方案高 1.9 倍.

综上, 在两种网络拓扑中, 分布式方案在较低内存空间下几乎能够始终保持最稳定且最高的大流检测精度, 而其他方案只能在增加内存 (如 100 kB) 后才能接近分布式方案在 15 kB 内存下的精度. 显然, 分布式方案能够实现最高的空间资源利用率, 在保证相同检测精度时所需的存储开销最低.

7 总结与讨论

本文提出了一种全新且高效地面向 SDN 网络的分布式轻量级大流检测方案, 能够检测 SDN 网络中的 top-k 流及过阈值流. 该方案的核心思想是利用网络中的所有交换机协同测量网络流量, 从而分摊测量任务带给交换机的负载, 极大降低了每台交换机所需的处理开销, 避免处于流量较高链路上的交换机测量开销过大的极端情况. 同时, 该方案中测量模块采取的映射机制能使其本身具有更低的存储需求, 且其对每个到达的数据包最多更新一次计数器, 实现了近似最优的处理速度, 而将大流流标签实时上传至控制器保存的方式也避免了流标签在交换机的存储资源占用. 从实验结果来看, 本文提出的分布式测量方案相比现有方法显著地提升了吞吐量, 且在仅占用非常少量内存的情况下也能够获得较高的检测精度, 满足了低计算和存储开销下的高精度检测需求.

此外, 若网络中某些层级 (如汇聚层) 内的交换机采取如堆叠或集群这样更加复杂的连接方式时, 本文的方案仍能具有最佳性能. 这是因为分布式方案的关键是将流的测量任务分摊给它路径上的多台交换机, 所以即使某条流的转发链路上的某些节点采取了堆叠或集群连接, 分布式方案同样能应用其上, 降低链路上所有执行测量任务的节点的处理开销.

然而, 本文的局限之处在于仅研究了 SDN 网络中的大流检测问题, 并未研究如何根据大流检测的结果来优化如负载均衡、拥塞控制、突发流量管理等核心网络功能的性能. 我们未来的研究方向是在本文分布式大流检测方案的基础上延伸, 设计一种能够实现如负载均衡、拥塞控制等不同网络优化目标的 SDN 网络大流重路由机制. 通过该机制, 我们能在满足网络资源限制的条件下, 根据不同的优化需求来为分布式方案检测到的大流重新部署 SDN 路径, 从而提升核心网络功能的性能.

参考文献

1 Kirkpatrick K. Software-defined networking. Commun ACM, 2013, 56: 16–19
2 Hong C Y, Kandula S, Mahajan R, et al. Achieving high utilization with software-driven WAN. In: Proceedings of the ACM SIGCOMM Computer Communication Review, HongKong, 2013. 15–26
3 Zhao G, Xu H, Fan J, et al. Achieving fine-grained flow management through hybrid rule placement in SDNs. IEEE Trans Parallel Distrib Syst, 2020, 32: 728–742
4 Nunes B A A, Mendonca M, Nguyen X N, et al. A survey of software-defined networking: past, present, and future of programmable networks. IEEE Commun Surv Tut, 2014, 16: 1617–1634
5 Benson T, Akella A, Maltz D A. Network traffic characteristics of data centers in the wild. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, Melbourne, 2010. 267–280
6 Roy P, Khan A, Alonso G. Augmented sketch: faster and more accurate stream processing. In: Proceedings of the International Conference on Management of Data, San Francisco, 2016. 1449–1463
7 Yang T, Zhang H, Li J, et al. HeavyKeeper: an accurate algorithm for finding top-k elephant flows. IEEE ACM Trans Netw, 2019, 27: 1845–1858
8 Cormode G, Muthukrishnan S. An improved data stream summary: the count-min sketch and its applications. J Algorithm, 2005, 55: 58–75
9 CAIDA: anonymized Internet traces 2019. https://catalog.caida.org/dataset/passive_2019_pcap
10 Yu X, Xu H, Yao D, et al. CountMax: a lightweight and cooperative sketch measurement for software-defined networks. IEEE ACM Trans Netw, 2018, 26: 2774–2786
11 Wang H, Xu H, Huang L, et al. Fast and accurate traffic measurement with hierarchical filtering. IEEE Trans Parallel Distrib Syst, 2020, 31: 2360–2374
12 Thomas B. Cisco Visual Networking Index (VNI) Complete Forecast Update, 2017–2022. Americas/EMEAR Cisco Knowledge Network (CKN) Presentation, 2018
13 Kandula S, Sengupta S, Greenberg A, et al. The nature of data center traffic: measurements & analysis. In: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, Chicago, 2009. 202–208

14 Xu H, Huang H, Chen S, et al. Achieving high scalability through hybrid switching in software-defined networking. IEEE ACM Trans Netw, 2018, 26: 618–632

15 Estan C, Varghese G. New directions in traffic measurement and accounting. In: Proceedings of the ACM SIGCOMM Computer Communication Review, Pittsburgh, 2002. 323–336

16 Charikar M, Chen K, Farach-Colton M. Finding frequent items in data streams. In: Proceedings of International Colloquium on Automata, Languages, and Programming, Malaga, 2002. 693–703

17 Schweller R, Li Z, Chen Y, et al. Reversible sketches: enabling monitoring and analysis over high-speed data streams. IEEE ACM Trans Netw, 2007, 15: 1059–1072

18 Li J Z, Li Z K, Xu Y F, et al. WavingSketch: an unbiased and generic sketch for finding top-k items in data streams. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020. 1574–1584

19 Yang T, Gong J Z, Zhang H W, et al. HeavyGuardian: separate and guard hot items in data streams. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, 2018. 2584–2593

20 Yang T, Jiang J, Liu P, et al. Elastic sketch: adaptive and fast network-wide measurements. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, Budapest, 2018. 561–575

21 Zhao B H, Li X, Tian B Y, et al. DHS: adaptive memory layout organization of sketch slots for fast and accurate data stream processing. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, Singapore, 2021. 2285–2293

22 Tang L, Huang Q, Lee P P C. MV-Sketch: a fast and compact invertible sketch for heavy flow detection in network data streams. In: Proceedings of IEEE Conference on Computer Communications, Paris, 2019. 2026–2034

23 Huang H, Sun Y E, Chen S G, et al. You can drop but you can't hide: k-persistent spread estimation in high-speed networks. In: Proceedings of IEEE Conference on Computer Communications, Honolulu, 2018. 1889–1897

24 Huang H, Sun Y E, Ma C, et al. An efficient K-persistent spread estimator for traffic measurement in high-speed networks. IEEE ACM Trans Netw, 2020, 28: 1463–1476

25 Huang H, Sun Y E, Ma C, et al. Spread estimation with non-duplicate sampling in high-speed networks. IEEE ACM Trans Netw, 2021, 29: 2073–2086

26 Du Y, Huang H, Sun Y E, et al. Self-adaptive sampling for network traffic measurement. In: Proceedings of IEEE Conference on Computer Communications, Vancouver, 2021. 1–10

27 Du Y, Huang H, Sun Y E, et al. Short-term memory sampling for spread measurement in high-speed networks. In: Proceedings of IEEE Conference on Computer Communications, London, 2022. 470–479

28 Xu H L, Chen S G, Ma Q P, et al. Lightweight flow distribution for collaborative traffic measurement in software defined networks. In: Proceedings of IEEE Conference on Computer Communications, Paris, 2019. 1108–1116

29 Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture. SIGCOMM Comput Commun Rev, 2008, 38: 63–74

30 Metwally A, Agrawal D, Abbadi A E. Efficient computation of frequent and top-k elements in data streams. In: Proceedings of the 10th International Conference on Database Theory, Berlin, 2005. 398–412

31 Bai L, Tian L Q, Chen C. Elephant flow detection algorithm for high speed networks based on flow sampling and LRU. Comput Appl Softw, 2016, 33: 111–115 [白磊, 田立勤, 陈超. 基于流抽样和 LRU 的高速网络大流检测算法. 计算机应用与软件, 2016, 33: 111–115]

32 Bai L, Chen C, Tian L Q. A TCBF_LRU algorithm for identifying and measuring elephant flows in high speed network flows. J Comput Res Dev, 2014, 51: 122–128 [白磊, 陈超, 田立勤. 基于 TCBF_LRU 的高速网络大流检测算法. 计算机研究与发展, 2014, 51: 122–128]

33 Zhou Y, Yang T, Jiang J, et al. Cold filter: a meta-framework for faster and more accurate stream processing. In: Proceedings of the International Conference on Management of Data, Houston, 2018. 741–756

34 Jia P, Wang P H, Zhao J Z, et al. LogLog filter: filtering cold items within a large range over high speed data streams. In: Proceedings of the IEEE ICDE, Chania, 2021. 804–815

35 Li Y P, Wang F Y, Yu X, et al. LadderFilter: filtering infrequent items with small memory and time overhead. In: Proceedings of the ACM SIGMOD, Seattle, 2023

36 Flajolet P, Martin G N. Probabilistic counting algorithms for data base applications. J Comput Syst Sci, 1985, 31: 182–209

37 Venkataraman S, Song D X, Gibbons P B, et al. New streaming algorithms for fast detection of superspreaders. In: Proceedings of the Network and Distributed System Security Symposium, San Diego, 2005
38 Goyal A, Daumé III H, Cormode G. Sketch algorithms for estimating point queries in NLP. In: Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, Jeju Island, 2012. 1093–1103

A distributed and lightweight elephant flow detection algorithm for software-defined networks

Jingjing ZHOU[1], He HUANG[1*], Yu-E SUN[2,3], Yang DU[1] & Boyu ZHANG[1]

1. *School of Computer Science and Technology, Soochow University, Suzhou* 215006*, China*;
2. *School of Rail Transportation, Soochow University, Suzhou* 215131*, China*;
3. *Key Laboratory of Embedded System and Service Computing* (*Tongji University*)*, Ministry of Education, Shanghai* 201804*, China*
* Corresponding author. E-mail: huangh@suda.edu.cn

Abstract Detecting elephant flows in software-defined networking (SDN) has broad applications in areas such as load balancing, anomaly detection, traffic engineering, and service quality management. In SDN, network traffic is usually measured by flow tables based on ternary content addressable memory (TCAM). However, using the flow table alone to measure network traffic can only store a small number of flows since TCAM is typically small and cannot identify all elephant flows from massive network streams. Therefore, to tackle this challenge, existing work uses an additional measurement module to support elephant flow detection, i.e., deploying a compact data structure (called a Sketch) on a single switch or a set of edge switches to measure traffic. However, running a Sketch on a switch will consume its scarce high-speed memory and computing resources. The tremendous measurement overhead caused by heavy network traffic may affect the performance of other network functions. To address this issue, this paper proposes a novel distributed lightweight elephant flow detection solution. The proposed solution allocates the measurement task to the switches and uses a lightweight Sketch to perform the measurement task, thereby balancing the network-wide measurement overhead and reducing each switch's computing overhead and memory usage effectively. Experimental results on real Internet traces demonstrate that the proposed solution has higher accuracy, lower computing overhead, and smaller memory usage compared to the state-of-the-art techniques.

Keywords software-defined networks, network traffic measurement, elephant flow detection, distributed measurement, Sketch