目录

[引言 1](#_Toc157944242)

[1 背景 3](#_Toc157944243)

[2 从软件定义的角度调查细粒度的测量 6](#_Toc157944244)

[2.1 操作层 8](#_Toc157944245)

[2.1.1 基于草图的方法 9](#_Toc157944246)

[2.1.2 其他方法 17](#_Toc157944247)

[2.2 控制层 18](#_Toc157944248)

[2.3 应用层 25](#_Toc157944249)

[3. 结论 30](#_Toc157944250)

[参考文献 31](#_Toc157944251)

插图清单

[图 1 CM Sketch的概况图[14] 6](#_Toc157944341)

[图 2 软件定义角度的细粒度测量的系统框架图 7](#_Toc157944342)

[图 3 测量系统的层级划分图 8](#_Toc157944343)

[图 4 Deltoids的查找过程[15] 10](#_Toc157944344)

[图 5 ASketch 架构[17] 11](#_Toc157944345)

[图 6 Count Braids计数器的结构[59] 12](#_Toc157944346)

[图 7 SketchVisor 架构图[53] 13](#_Toc157944347)

[图 8 NitroSketch 框架的草图结构[61] 14](#_Toc157944348)

[图 9 DREAM 系统框架图[35] 18](#_Toc157944349)

[图 10 OpenSketch架构图[36] 21](#_Toc157944350)

[图 11 SCREAM框架图[55] 22](#_Toc157944351)

[图 12 SCREAM框架图[55]. 23](#_Toc157944352)

[图 13 Frenetic查询的语法[110] 26](#_Toc157944353)

[图 14 Path Query 的语法[111] 27](#_Toc157944354)

[图 15 Sonata的概况[111] 28](#_Toc157944355)

附表清单

[表格 1操作层基于草图的总结 15](#_Toc157944375)

[表格 2 控制层测量系统的总结 24](#_Toc157944376)

[表格 3 Marple语言功能构造的总结 28](#_Toc157944377)

[表格 4 测量语言的对比 29](#_Toc157944378)

# 引言

随着网络使用和流量的激增，由于网络配置不佳和恶意攻击，拥塞和异常情况变得更加常见，这会危及网络的稳定性和安全性，导致经济损失。为了保证网络性能，网络测量提供了一种有效的方式来从数据平面收集探测消息，从而有助于检测瓶颈和异常行为。根据最近的一项研究[1]，到2020年，网络流量监控和分析市场的收入预计将达到23.2亿美元，年复合增长率为24.7%。近来，像Azure[2]和Google Cloud[3]这样的主流云服务提供商已经开发出更高效和精细的网络测量工具，提高了网络的透明度，降低了性能分析和网络优化的门槛。

网络测量的研究始于上世纪80年代。网络管理员根据具体的测量任务定义监测范围和粒度，而某些测量任务（如异常检测、性能诊断）需要对网络流量进行细粒度测量。然而，计算设备和交换机的硬件限制给处理所有流量数据带来了巨大挑战。因此，网络管理员不得不对这些数据包进行采样，并仅存储采样部分的信息。这些基于采样的测量方法由于存储和计算开销较低而被广泛应用，例如NetFlow[4]、sFlow[5]、sticky sampling[6]、sample and hold[7]、OpenWatch[8]和[9]、[10]。然而，这些算法会丢失大量的流量信息，因此无法保证测量结果的准确性。因此，很难检测瞬时异常和计算流量分布。

在现代数据中心环境中，大量的网络流量需要进行有效的管理[11]，[12]，[13]，这严重依赖于高分辨率和细粒度的网络测量方法。目前学者们已经提出了一些利用这些资源来实现分组级监测的方法[14]，[15]，[16]，[17]，[18]。这些方法利用设计良好的数据结构或先进的硬件功能来解决与网络状态和流量条件相关的一系列查询，从而捕捉到网络管理员感兴趣的完整情况。

最近，研究人员通过具备特定需求和改进算法的专用网络设备（例如[15]，[17]，[19]）来提升网络测量的性能。然而，供应商特定的ASIC无法支持定制化的数据包处理流水线，这阻碍了为细粒度网络测量设计的先进数据结构的部署。可编程数据包处理器（P4[20]）和虚拟交换机（例如Open vSwitch[21]，BESS[22]）的出现为新的测量方法设计带来了更多选择。例如，结合P4语言和最新的可编程以太网交换机[23]，可以以每秒12.8 Tbps的速度实现灵活性。此外，虚拟/软件交换机广泛部署在虚拟机和容器环境中，用于支持虚拟机或容器的数据包转发。由于软件的灵活性，可以在网络测量中实现更复杂的过程，并无缝地融入虚拟交换机。P4和虚拟交换机为交换机提供了额外的计算能力，并共同努力实现细粒度的网络测量。

在研究界中，已经提出了许多新颖的方法和系统来进行高速网络的网络测量。根据测量粒度的不同，我们将这些方法和系统分为两类：

* 粗粒度测量方法：一些测量方法，如NetFlow[4]和sFlow[5]，采用采样策略来适应流量速度。它们记录每N个数据包中的一个数据包，采样率可以由网络管理员进行定制。采样的数据包由远程收集器收集，并由上层应用程序进一步分析[24]，[25]，[26]。当发生流量突发时，交换机通过降低采样率来处理增加的流量规模[27]，但也更有可能错过许多小流量。尽管已经有很多研究工作致力于从采样部分推断出原始流量[28]，[29]，[30]，但由于需要保证测量操作可以负担，采样率较低，因此准确性仍然有限[31]。我们将这种方法定义为粗粒度方法，因为它们丢失了太多的流量信息，特别是对于小流量，这对于许多测量任务来说是重要的。
* 细粒度测量方法：与基于采样的方法相比，一些数据流算法对所有数据包进行单次遍历，然后生成流量特征的估计值。例如，基于草图的算法（如Count Sketch[32]、CU Sketch[7]、Count-Min Sketch[14]）使用草图对所有观测到的数据包的流量统计进行总结，这是对原始流量数据的一种有损压缩形式。虽然总结过程会导致分析的分辨率损失（由于哈希碰撞），但错误程度在理论上与可用资源的数量相关。除了基于草图的方法，还有许多其他方法也可以用于细粒度网络测量（如Frequent[33]、有损计数[34]）。我们将这些方法定义为细粒度测量方法，因为它们能够对网络流量进行计数和总结，而不会丢掉任何流量数据包。

在本次调查中，我们明确关注细粒度的网络测量方法，因此粗粒度的方法超出了我们的研究范围，因为在高流量场景下它们会丢失太多的信息。需要注意的是，一些算法（例如Count-Min Sketch[14]）并非专门为网络测量而设计，但它们在当前的细粒度测量系统中扮演着重要的角色[35]，[36]。

我们认为，理想的网络测量系统应具备通用性、灵活性和用户友好性。尽管现有的细粒度方法已经满足了网络测量的效率和准确性要求，但仍存在一些挑战。首先的挑战是大多数方法只支持特定的测量目标。例如，Frequent[33]只能找出所有的主要项，而无法准确计算频率。像Count-Min Sketch[14]这样的基于草图的算法则无法获取流的基数[37]。更重要的是，大多数测量方法并未考虑可用性。例如，基于草图的算法在实际部署中需要大量手动配置参数[38]。此外，一些测量目标（例如重要流量检测[6]）是基于阈值的。然而，现有的重要流量检测方法将阈值作为输入，而测量准确性与阈值的选择密切相关[38]。在本次调查中，我们将细粒度的网络测量抽象为软件定义架构。具体来说，我们将测量系统划分为三个层级：操作层、控制层和应用层。我们讨论了每个层级的职责，并将现有的细粒度测量方法和系统分类到这个架构中。像草图这样的数据流方法属于操作层，控制层则协调现有的测量数据结构和算法来执行特定的测量任务。在应用层中，网络管理员可以通过用户友好的API定义测量应用，而无需考虑使用哪种具体的方法。

需要注意的是，本报告并未在设计或实现方面对软件定义测量架构进行全面介绍。我们的重点在于对细粒度测量的各种方法和系统进行系统调查，并从软件定义的角度对其进行分类。我们希望通过这次调查和观点，能够全面了解这一领域，并激发出更多有关细粒度网络测量的新思路。

现有文献中的大多数相关调查都集中在数据分析方面。Alconzo等人[39]主要回顾了大数据分析中的高容量测量数据。调查[40]也侧重于大数据分析，但主要讨论网络入侵检测。我们的调查与[41]不同，后者介绍了集成网络管理工具，管理员可以直接通过图形界面操作来查看流量信息。据我们所知，这是关于高速网络细粒度测量的首次调查。

我们在第二节开始时介绍了细粒度测量的背景知识、一些基本测量目标、三个基本算法以及我们对相关工作选择的原则。然后，我们将现有的细粒度测量方法和系统分别归类到操作层（见第2.1节）、控制层（见第2.2节）和应用层（见第2.3节）。最后，在第三节中总结了全文，并讨论了细粒度网络测量的未来发展方向。

# 1 背景

在本节中，我们将简要介绍细粒度测量的背景知识。首先，我们给出了一些在本报告中广泛使用的基本术语和测量目标的定义。然后，我们介绍了对当前网络测量系统产生深远影响的三个基本算法。

在网络测量中，我们将收集信息的时间段定义为一个时期。考虑到存储能力的限制，交换机通常需要在短时间内记录流量信息，然后将信息发送到服务器，以便交换机可以清除旧记录，为下一个小时间段腾出空间。这个小时间段被称为时期，或者称为测量时期。此外，当流的数据量明显大于大多数其他流，并且至少包含个数据包时，我们将其称为重流或者大流量。相反，当流的数据包少于个时，我们将其称为轻流或者小流量。常数是根据不同的流量特征而变化的参数[42]。

测量目标是我们在测量任务中希望了解的具体指标。在这里，我们列出了几个最基本的测量目标。

重要流（Heavy hitter）[6]：在测量时期内总大小超过特定阈值的流量。重要流量检测可以帮助管理员了解在一个时期内哪些流量占用了最多的网络带宽。

Top-k[43]：在测量时期内具有最大大小的k个流量。它可以用于分析哪些网页最受欢迎，或者在一个时期内哪些流量占用了最多的带宽。

分层重要流（HHH）[44]：由最长前缀（例如源IP的前缀）定义的流量，在排除任何分层重要流量的后，其总大小超过了特定阈值。例如，我们使用32位源IP来定义流量，然后可以构建一个32级二叉树，其中叶节点是真实的IP地址，其他节点是像192:168:0:X这样的前缀。该前缀流量的大小是所有前缀为192:168:0:X的流量的总和。如果192:168:0:X的所有子流量都很小，但是192:168:0:X的大小超过了阈值，则前缀流量192:168:0:X被视为分层重要流量。如果前缀192:168:X:X的大小在减去所有分层重要流量的后超过了阈值，那么该前缀就是一个分层重要流量。从集群的角度来看，分层重要流量可以被视为重要流量。

重要变化（Heavy changer，HC）[45]：在两个连续测量时期之间，大小差异超过特定阈值的流量。重要变化者可以被视为拥塞或恶意攻击的信号。

超级传播者（Superspreader，SS）[46]：在测量时期内向超过特定阈值数量的目标主机发送数据的源主机。它可以是用于检测主机可能存在的漏洞的网络扫描器。

DDoS[47]：在测量时期内从超过特定阈值数量的源主机接收数据的目标主机，表示目标主机正在遭受DDoS攻击。

基数（Cardinality）[37]：测量时期内不同流量的数量。它可以用于计算某个服务或服务器的访问频率。

流量大小[48]：任何可以由用户定义的流键识别的流量的大小。这对于网络服务提供商来计算用户产生的流量成本很有帮助。

流量大小分布[31]：测量时期内流量大小的分布。该目标旨在分析整体带宽占用情况，而不仅仅关注单个流量，这对于改善网络性能很有帮助。

上述所有的测量目标都需要对网络流量进行细粒度的时变测量。然而，流量速度的迅速持续增长给这些目标的测量带来了巨大挑战。幸运的是，数据流技术已经被引入以进行近似测量，它在准确性、速度和内存使用之间取得了平衡。在这里，我们介绍三种具有深远影响的代表性算法，它们对当前的网络测量系统产生了重要影响。

有损计数（Lossy Counting）[6]的目标是找出频率超过用户定义阈值的所有项。给定一个非常小的数，用于找出频率超过的项，该算法将输入流按照概念上的宽度划分为多个桶，桶的编号从1到变化。算法维护一个数据结构，其中表示流中的输入项，表示估计的频率，表示的最大误差。在处理拥有总处理包数的当前输入项时，如果已经存在，则将增加1，否则创建一个新的元组。满足的元组将被剔除，因为他们的频率的上届小于。在[34]中，每当开始处理一个新的桶时，将所有元组的减1，然后删除估计频率的元组。该算法的空间利用率为。有损计数可以找出网络测量中超过整体流量一定比例的重要项，并提供有界的估计。

布隆过滤器（Bloom Filters）[49]用于检查数据包是否属于已存在的流量，或者检查一个流量是否是已知重要项的成员。最经典的布隆过滤器[49]是在1970年提出的，并且被广泛用于动态测试一个项是否已经存在。在布隆过滤器中，需要一个初始全零的二进制数组和若干个哈希函数。要将一个项添加为成员，经典布隆过滤器需要使用这个哈希函数对项i进行哈希，找到数组中的个位，并将这些位设置为1。要测试一个项是否是成员，同样使用这个哈希函数找到映射位并进行检查。如果所有的个位都是1，布隆过滤器认为它是一个已存在的成员；如果这个位中的任何一位是0，布隆过滤器确定该项不是成员。由于哈希碰撞的存在，该算法可能会导致误报。在实践中，布隆过滤器通常与其他流量测量算法协同使用，以提高测量任务的效率和准确性[50]。

Count-Min Sketch（CM Sketch）[14]是当今最受欢迎的基于草图的算法之一，用于找出重要项。它是一种过估计算法，但每个计数器只需要一次内存访问。CM Sketch由一个大小为的计数器数组表示，其中个哈希函数的作用范围从到。如图1所示，对于每个具有数量的输入项，CM Sketch将添加到所有行的计数器中。频率估计也等于中的最小计数。设置和可以确保的概率至少为。CM Sketch的更新时间为，空间复杂度为。CM Sketch具有简单高效的操作、较小的内存使用和高准确性保证，因此成为网络测量中流行的流量计数算法。

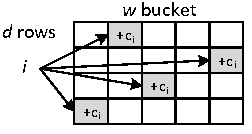


图 1 CM Sketch的概况图[14]

回顾一下，我们将网络测量方法分为两大类。上述的三种算法可以看作是细粒度测量方法，因为它们可以检查或记录所有经过的数据包。虽然它们在网络测量中并不是独一无二的，但它们构成了一个重要的基础，许多细粒度测量方法和系统都是基于它们构建的（例如[16]、[17]、[35]、[36]、[52]、[53]）。然而，布隆过滤器的功能有限，而仅使用像有损计数这样的计数器并不够通用因为它仍然会丢失大量关于流量的信息。例如，有损计数[6]无法获得流的基数[37]，因为它只保留了频率超过特定阈值的流。与有损计数相比，草图更适用于不同类型的测量目标，因为它们记录了所有被监测流的容量。

相关工作选择的原则。研究界已经积累了大量关于网络测量的研究工作，因此必须进行选择，以便突出主题的清晰性并考虑空间限制。我们选择论文的原则可以总结如下：

* 我们优先考虑在知名会议和期刊上发表的论文，以及这些论文中引用的高度相关的工作。
* 本文仅讨论细粒度测量方法，不包括粗粒度方法，因为它们无法适应快速发展的流量变化。
* 我们的讨论范围不仅限于测量算法，还包括对细粒度测量系统的其他研究，例如测量语言[54]、多任务管理[55]和测量数据收集[56]。
* 出于时效性的考虑，我们倾向于选择过去五年内与网络测量相关的论文。

在接下来的部分中，我们将从软件定义的角度对所选论文进行全面总结并进行讨论。

# 2 从软件定义的角度调查细粒度的测量

在本节中，我们将讨论一些最近提出的用于进行细粒度测量的高级数据结构、算法和系统。首先，我们需要介绍我们调查的视角。我们认为，一个成熟的测量系统需要具备通用性、灵活性和用户友好性。参考软件定义网络（SDN）的思想，我们将细粒度测量系统抽象为三层架构，包括操作层、控制层和应用层。如图2所示，我们通过将现有的细粒度测量方法或系统按照这三层架构进行分类来组织本节内容。

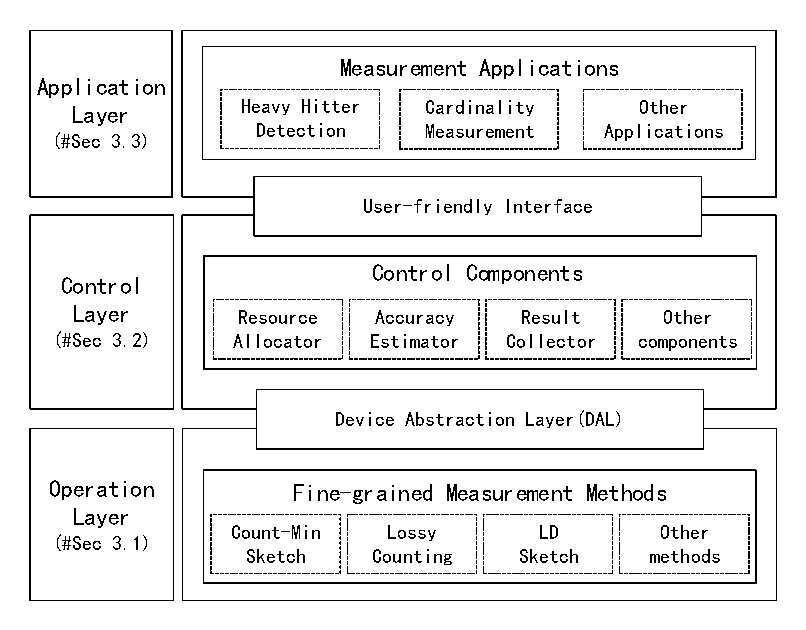


图 2 软件定义角度的细粒度测量的系统框架图

操作层：在操作层中，特定的细粒度测量方法被安装在不同网络设备的数据平面上。它们收集每个经过的数据包的记录，以便我们可以通过查询相应的数据结构来清楚地了解当前的流量特征。在第2.1节中，我们对最近提出的用于进行细粒度网络测量的高级数据结构和算法进行了全面的讨论。

控制层：在控制层中，测量控制器选择适当的数据结构和算法来满足不同测量应用的需求。然后，这些应用的需求可以看作是控制器调度的多个测量任务。控制器负责在硬件或软件设备中安装具体的细粒度测量方法，并在资源占用和测量精度之间进行权衡。一个成熟的测量系统还需要一个设备抽象层（Device Abstraction Layer，DAL），以便在不考虑使用哪种设备类型的情况下透明地安装测量方法。此外，由于大多数测量系统使用分布式测量架构（例如[48]、[57]），测量数据是在控制层中收集的。在多任务场景中，控制器还需要调度和管理多个测量任务。因此，在控制层中有几个控制组件（例如，资源分配器、结果收集器）。在第2.2节中，我们讨论了几个细粒度测量系统，它们提出了新的机制来管理测量方法和调度测量任务。

应用层：传统情况下，网络管理员需要根据特定的管理任务手动进行网络测量，这是耗时且容易出错的。从软件定义的角度来看，这个过程可以抽象为在应用层上运行的测量应用。网络管理员通过用户友好的界面定义几个测量应用，以便自动进行测量和收集结果。在第2.3节中，我们介绍了几个测量接口，并讨论了对细粒度网络测量接口的需求。需要注意的是，具体的测量应用不在我们讨论的范围之内。

对于每一层，我们将详细讨论几个最先进的工作。然而，有些测量系统不仅仅专注于单一层面。我们在图3中展示了这种现象。例如，OpenSketch[36]不仅提供了一个新颖的数据平面设计来实现不同的草图，还通过一个草图管理器和资源分配器自动管理草图，并为管理员提供了用户友好的API来定义特定的测量任务。这意味着OpenSketch涵盖了所有三个层面。在本调查中，我们使用OpenSketch（C）表示OpenSketch的控制层，而OpenSketch（O）和OpenSketch（A）分别定义为OpenSketch的操作层和应用层。其他多层测量系统也以同样的方式进行讨论。我们仔细讨论了这些多层系统的不同层面特点。大多数情况下，它们不能单独讨论，因为各层之间密切相关[53]，[54]，[58]。在接下来的三个小节中，我们将从下到上详细讨论最近的细粒度测量方法和系统。

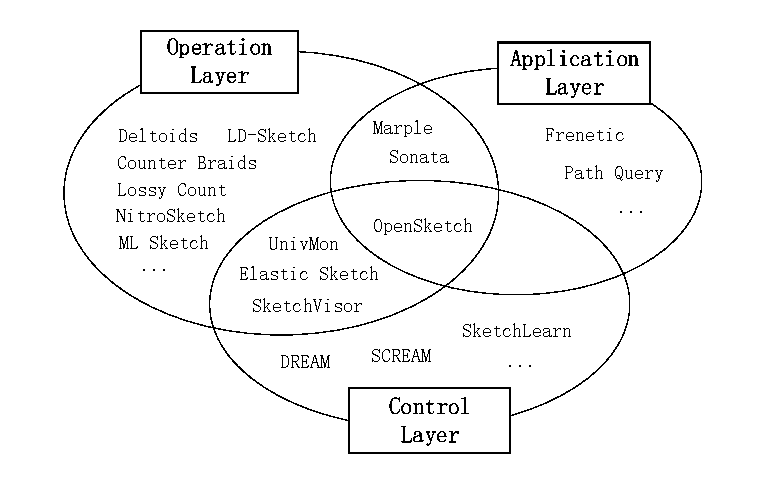


图 3 测量系统的层级划分图

## 2.1 操作层

在操作层中，不同设备上运行具体的数据结构和算法，执行各种测量任务。其中，基于草图的测量方法因其多功能性、高性能和易于部署而受到广泛研究。为了进行深入讨论，我们将细粒度测量方法大致分为两类：基于草图的方法和其他方法（即非草图方法）。

### 2.1.1 基于草图的方法

回顾我们在第1节中简要介绍的Count-Min Sketch，它是目前最受欢迎的用于进行细粒度测量的草图算法之一。许多测量方法和系统都基于它（例如[15]、[18]、[48]）。然而，Count-Min Sketch也存在一些局限性。现在我们列出了主要的四个局限性：

* 局限性Ⅰ：缺乏流关键字记录。在实际的生产环境中，存在大量的流量，记录所有的流关键字是一个巨大的挑战[15]，[16]，[52]。Count-Min Sketch本身并不记录任何流关键字，用户需要提供候选流键来检查频率，这限制了草图的通用性。
* 局限性Ⅱ：资源冲突导致的精度损失。在使用有限资源跟踪大量网络流量时，大流量和小流量之间的哈希碰撞频繁发生，导致严重的精度损失[38]。
* 局限性Ⅲ：固定的计数器大小。Count-Min Sketch中的所有计数器都具有固定的长度和固定的计数容量。当计数器用于计数大流量时，可能会发生溢出。而当计数器用于计数小流量时，该计数器的高位将被浪费[59]，[60]。
* 局限性Ⅳ：显著的计算开销。草图需要为每个数据包执行多次哈希计算，这会引入显著的计算开销，尤其是对于软件交换机[53]，[61]，[62]。

在这里，我们详细讨论几种具有代表性的测量算法或系统，每种算法或系统都解决了上述一种或多种限制。由于篇幅限制，其他使用类似技术的方法将只能简要讨论。

Deltoids[15]旨在寻找不同时间段或切换之间的重大变化，并能够恢复重大变化的流量关键信息，从而间接解决了局限性Ⅰ。Deltoids基于分组测试技术，分为两个部分：识别和验证。识别部分找出可能是重大变化的候选流量，并将其流量关键信息恢复出来。验证部分则从候选集中排除非真正的重大变化项。Deltoids的工作流程如图4所示。识别结构是一个扩展的CM Sketch，需要个存储桶和个哈希函数，将映射到。每个存储桶代表一个项组，其中包含了所有映射到该桶的项。在实践中，每个存储桶还保留了个计数器结构，用于计算在中以二进制表示的第个位置为1的项的数量。特别地，计算了中的所有项。验证结构需要个存储桶和个哈希函数，将映射到，每个存储桶也代表一个具有单个计数器的组。在更新过程中，对于映射到项的每个存储桶，如果在第个位置的二进制表示为1，Deltoids将更新所有计数器的数量。然后，Deltoids直接将项的数量更新到验证的草图中。通过结构估计重大变化，并使用进行验证。查找绝对和相对Deltoids所需的空间复杂度为，更新时间复杂度为。Deltoids通过从识别部分恢复流量关键信息来解决局限一。识别中的草图将CM Sketch的每个存储桶从一个单一计数器扩展为多个，这会带来更多的内存开销。此外，每个存储桶中的计数器数量与数据包总数有关，这意味着用户需要事先了解要监测的流量关键信息的规模。

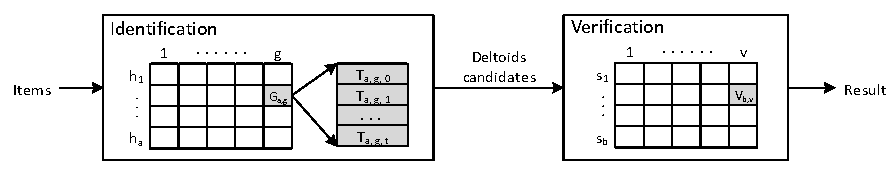


图 4 Deltoids的查找过程[15]

增强型草图（ASketch）[17]是一种流处理框架，利用分离计数结构来提高估计准确性。它解决了Count-Min Sketch中的局限性Ⅱ，即小流量可能与大流量发生碰撞，导致计数不准确。ASketch包含两个数据结构：过滤器和草图。如图5所示，过滤器用于存储频繁项，每个项包含和。对于流中的每个项，其值为，ASketch首先在过滤器中查找。如果项已存在于过滤器中，则只增加。如果项不存在于过滤器且过滤器未满，它将被插入到过滤器中，设置为0，设置为。当过滤器已满时，项将被发送到草图执行正常的更新过程。将项插入草图后，ASketch将当前项的估计频率与过滤器中最小的进行比较。如果大于最小的，ASketch会交换这两个项。项将移至过滤器，并同时设置和为。被移除的项，其最小，在插入到草图时使用更新值，因为已经包含在上次从草图中移除该项时更新。请注意，过滤器和草图之间的交换只会触发一次。也就是说，在插入到草图后，如果被移除项的估计频率大于过滤器中的最小，将不会再触发其他项的交换。ASketch使用元组来记录潜在的重要项，所有的元组都存储在过滤器中。在真实的数据集下，由于网络流量的偏斜特性，即20%的大流量占据了整体流量的80%，ASketch相比单个草图可以实现30%至40%甚至更高的数据流处理吞吐量，并且提高了估计的准确性，这是因为主要的大流只访问过滤器，然后进行更新，只有20%的流量会访问第二个草图。这种分离有效减少了大流量和小流量之间的哈希冲突，并且过滤器还记录了大流量的流量关键信息，这意味着ASketch也能解决局限性Ⅰ。

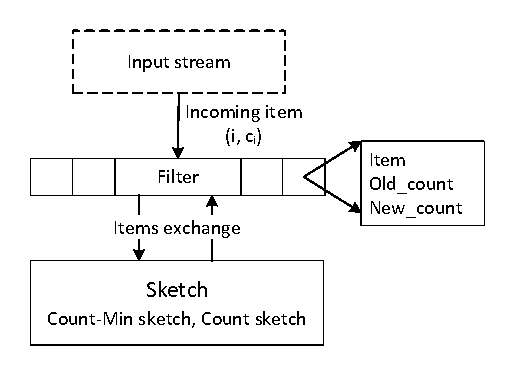


图 5 ASketch 架构[17]

Counter Braids[59]是一种使用分层计数器的结构，用于解决局限性Ⅲ：草图中的所有计数器都具有固定长度和固定计数容量。该结构的主要目标是为小流量节省计数器位，并使用更多的分层计数器记录大流量。Counter Braids使用基于图的低复杂度算法，在测量周期结束时以几乎零误差的方式重构流量大小。Counter Braids具有分层结构，其中第层具有个计数器，每个计数器的深度为位，如图6所示。假设总层数为。在每一层中，计数器的数量可以递减，因为更高的层意味着更大的流量大小，而网络中的大流量数量是有限的。低层的计数器包含一个额外的状态位，初始值为0，在计数器溢出后设置为1。Counter Braids仅使用一组哈希函数将流映射到第一层计数器，然后在连续的两个层之间进行映射。对于每个输入项，它被哈希到的所有第一层计数器都会递增。如果计数器溢出，那么哈希到的下一层中的所有计数器都会递增，并且计数器的值将被重置为0，并将其状态位设置为1。重构算法从最高层到最低层解码这些计数器，并在每个测量周期结束时计算每个流的大小。在解码过程中，消息（计算的计数器值）通过连续层之间的映射关系迭代传递，然后可以计算流的大小[59]。请注意，解码过程仅在每个周期结束时执行，并且它解码的是所有流的大小，而不仅仅是重要的流。因此，Counter Braids不是一个即时的函数，但它可以恢复它记录的所有流的大小，这对于测量实际网络中的流量大小和流量大小分布非常有用。

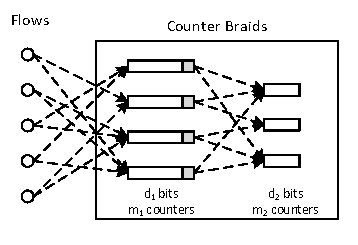


图 6 Count Braids计数器的结构[59]

SketchVisor(O)是SketchVisor[53]的操作层，它是一种用于高速软件包处理的强大网络测量框架。如图7所示，SketchVisor(O)分为两个部分：正常路径和快速路径。正常路径部署了一个或多个基于草图的解决方案，并处理来自有界FIFO缓冲区的数据包。当流量负载超过正常路径的处理能力，且FIFO缓冲区变满时，SketchVisor(O)将溢出的数据包重定向到快速路径，该路径部署了一种新的低计算开销的top-k算法。在快速路径中，每个流与哈希表中的三个计数器相关联。计数可以在插入流之前丢失的最大字节数，计数流插入后剩余的字节数，计数插入后递减的字节数。快速路径维护一个变量，用于计算所有递减字节数的总和。与常规基于计数器的测量算法相同，当哈希表未满时，首次出现的大小为的流将被插入到哈希表中，其计数为。如果已经存在于哈希表中，SketchVisor(O)只增加相应的计数器。否则，算法将计算递减值，哈希表中的每个流计数器减少，相应增加。计数器递减后，其计数器不大于0的流将从哈希表中删除。如果，则将添加为。快速路径在数据包处理过程中丢失了一些信息，因此比正常路径的精度较低。为了实现更准确的测量，正常路径应尽可能处理尽可能多的数据包，而快速路径仅在必要时激活。SketchVisor(O)可以通过使用双路径模式来处理不同的流量速度（例如高达10Gbps[53]）。双路径模式会导致一个问题，部分数据包将被完整处理，而其他数据包则不会。然后，我们认为SketchVisor(O)会导致信息丢失，即并非所有数据包都执行相同的计数过程。幸运的是，SketchVisor的控制层通过采用恢复算法来缓解这个问题，该算法使用矩阵插值和压缩感知来消除由于快速路径处理而产生的额外误差。

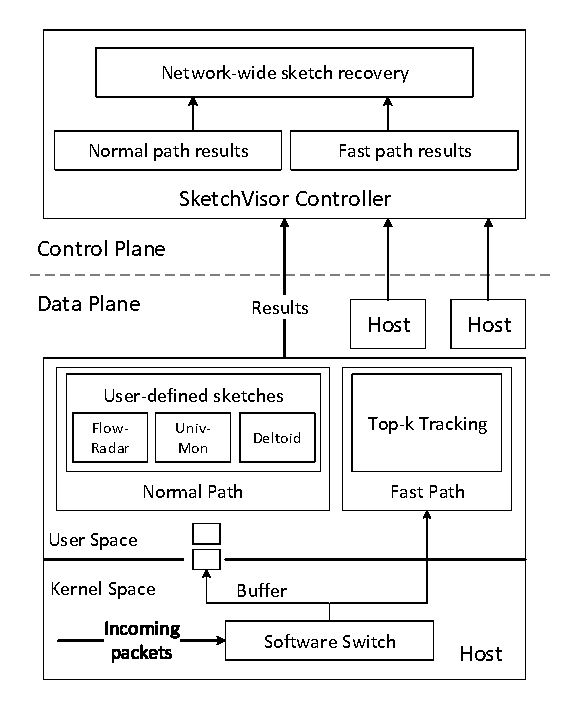


图 7 SketchVisor 架构图[53]

NitroSketch[61]是一个针对现有基于草图的算法的外部框架，旨在降低软件交换机中测量算法的计算开销（局限性Ⅳ）。NitroSketch的设计基于一个关键属性，即软件交换机具有更多的存储资源，而计算资源成为瓶颈，这与硬件交换机相反。在软件交换机中，草图算法的关键瓶颈包括每个数据包的多次昂贵的哈希计算、多个随机内存访问和更新以及跟踪top-k个流的额外开销。NitroSketch提出了三个关键思想。首先，它通过对多个独立计数器数组进行采样，而不是对每个数据包进行采样。在处理过程中，NitroSketch根据采样率在每个数组中“抛硬币”，决定是否需要在该数组中更新计数器，从而减少了哈希和计数器更新的次数。其次，NitroSketch改进了“抛硬币”的方式，采用几何采样方法进一步降低计算量。传统的逐行硬币抛掷方式会引入大量计算开销，而NitroSketch则采用几何分布采样，直接决定下一个要更新的计数器数组以及在此之前要跳过多少个数据包。第三，为了保证适当的收敛时间，NitroSketch根据数据包到达率自适应调整采样率。通过跳过更多的数据包，可以保证准确性，但也会导致较长的等待时间。因此，在数据包到达率较低时，可以增大采样率以记录更多数据包。NitroSketch提供了两种自适应采样模式：1) AlwaysLineRate模式，动态将p设置为与当前数据包到达率成反比；2) AlwaysCorrect模式，初始p设为1.0，并在满足收敛条件后切换到AlwaysLineRate模式。图8展示了NitroSketch框架的整体视图。当数据包到达软件交换机时，NitroSketch选择要更新的数据包和计数器数组，从而跳过了大部分数据包。采样的数据包根据不同草图算法的更新函数来更新选定的计数器数组（例如，在图8中，第一个数据包更新了Array 2中的一个计数器，然后NitroSketch选择下一个要更新的数组，即当前数组之后的第21个数组，然后跳过中间的4个数据包，最后一个数据包将更新Array 3中的计数器）。使用NitroSketch，管理员可以根据实际需求选择采样模式。该框架具有通用性，可应用于任何现有的草图结构，以提高数据包处理性能，同时保证理论上的精度。因此，尽管采用了采样技术，NitroSketch仍能提供精细的测量，可被视为现有系统的补充，以实现高线速处理。

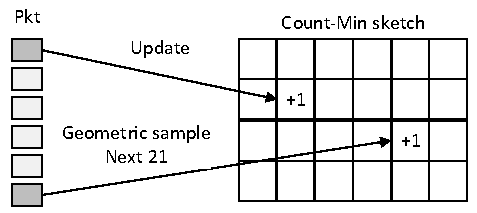


图 8 NitroSketch 框架的草图结构[61]

这些基于草图的方法使用了几种代表性的技术来解决上述的限制。还有许多其他类似的草图方法，它们在解决其他特定问题时有一些创新点。例如，SketchLearn[38]不仅采用了与上述方法类似的技术来解决数据平面（SketchLearn(O)）中的局限性Ⅱ和局限性Ⅲ，还减轻了网络管理员的负担，并支持对收集的信息进行扩展查询。由于篇幅限制，我们在这里简要介绍其他一些草图方法。Count-Min-Heap[63]使用堆来跟踪所有候选的重要流量（即可以解决局限性Ⅰ）及其估计的总和。MV-Sketch[64]是一种可逆草图（即可以解决局限性Ⅰ），支持重要流量和重要变化的检测，并可用于分布式检测。FlowRadar[65]提出了编码流集，这是Invertible Bloom Filter Lookup Table（IBLT）的扩展。对于每个存储桶，FlowRadar保留了数据包计数、流量数量和该存储桶中映射的所有不同流键的异或值。此外，通过对编码的流集执行解码算法，FlowRadar可以获取所有的流键。SpreadSketch[66]也是一种可逆草图，但其目标是在整个网络范围内执行超级传播者（superspreader）检测。Reversible Sketch[16]通过使用可逆哈希函数来恢复流键，从而解决了局限性Ⅰ。类似于可逆草图，SeqHash[67]将键的单词哈希到草图中，并从重要的存储桶中反转重要的键。SeqSkech和EmbedSketch是在[68]中介绍的，通过正确结合压缩感知理论和草图算法，几乎可以达到零误差。Cold Filter[18]是一种类似于Augmented Sketch[17]的分离计数算法，但它只需要过滤器和草图之间的单向通信。它还旨在提高测量的准确性（即可以解决局限性Ⅱ）。Pyramid Sketch[60]通过使用金字塔形的数据结构，根据当前传入项的频率自动扩大相应计数器的大小，从而解决了局限性Ⅱ和局限性Ⅲ，这与Count Braids[59]类似，但更高效。Lossy Conservative Update（LCU）sketch[69]在CUsketch[7]的基础上结合了有损计数的概念（详见第1节），它将流分为窗口，并在窗口边界处递减草图计数器，以减少过度估计的误差。Elastic Sketch(O)[48]也像Augmented Sketch一样将大流量与其他流量分离，并且可以根据带宽、数据包速率和流量大小分布进行自适应调整。LD-Sketch[52]是一种数组式的草图结构，它结合了基于计数器的技术来解决Count-Min Sketch无法记录任何流键的局限性局限性Ⅰ。它设计用于准确检测重要流量和重要变化，并采用分布式架构。UnivMon(O)[57]使用通用流式处理[70]，其中单个通用草图可以准确估计大类函数。Defeat[71]将草图与子空间方法[72]结合起来检测异常，并可以确定导致异常的IP流量。快速草图[73]以更高的效率和可靠性将数据包聚合到少数几个流中，并可用于查找重要变化。TCM[74]使用一组图形草图来总结流量并监视流，这些草图的尺寸远小于原始图形，然后可以支持更多类型的图形分析（例如，流路径）。偏差感知草图[Bias-Aware Sketches，75]提出了偏差感知的线性草图，严格推广了在偏差数据流下的标准草图的误差保证。ML Sketch[76]通过将机器学习理论引入网络测量中，提高了估计准确性（局限性Ⅱ）。奇数草图[Odd sketch，77]是一种紧凑的二进制草图，用于估计两个集合的Jaccard相似度。

**草图方法的结论**：作为所有监视流量的有损压缩，草图可以支持广泛的测量目标。因此，我们可以发现大多数基于草图的方法通过利用草图的普适性来获得广泛的测量目标。这些基于草图的算法旨在使草图更加通用、准确和高效。我们在表1中总结了这些方法及其创新点。

表格 1操作层基于草图的总结

|  |  |  |  |
| --- | --- | --- | --- |
| 测量方法 | 解决的局限 | 创新点 | 测量目标 |
| Deltoids[15] | Ⅰ | 恢复候选流键 | 重要流，重要变动，流大小 |
| Augmented Sketch[17] | Ⅰ，Ⅱ | 记录部分流量键提高估计精度 | 重要流，重要变动，流大小 |
| FlowRadar[65] | Ⅰ | 解码编码流集中的所有流键以提供通用性 | 通用目标 |
| Counter Braids[59] | Ⅲ | 提供分层计数器并最小化流量计数中的内存使用。 | 重要流，重要变动，流大小 |
| SketchVisor(O)[53] | Ⅳ | 引入快速路径以加速处理速度。 | 通用目标 |
| NitroSketch[61] | Ⅳ | 在草图上进行几何分布抽样 | 通用目标 |
| Count-Min-Heap[63] | Ⅰ | 记录部分流键 | 重要流，重要变动，Top-k，流大小 |
| MV-Sketch[64] | Ⅰ | 返回候选流键 | 流量检测 |
| SeqSkech and EmbedSketch[68] | Ⅰ，Ⅱ | 通过引入压缩感知，以达到近乎零误差的结果。 | 通用目标 |
| SpreadSketch[66] | Ⅰ | 返回候选流键 | 超级传播者 |
| Reversible Sketch[16] | Ⅰ | 通过可逆哈希来恢复候选流键。 | 重要流，重要变动，流大小 |
| SeqHash[67] | Ⅰ | 通过一种新颖的顺序哈希方案来恢复候选流键。 | 重要流，重要变动，流大小 |
| Cold Filter[18] | Ⅰ，Ⅱ | 记录部分流键并提高估计准确性。 | 重要流，重要变动，Top-k，流大小 |
| Pyramid Sketch[60] | Ⅲ | 使用金字塔型数据结构并最小化内存使用。 | 重要流，重要变动，Top-k，流大小 |
| LCU sketch[69] | Ⅱ | 结合有损计数的思想，进一步减少产生的过估计误差。 | 重要流，重要变动，流大小 |
| Elastic Sketch(O)[48] | Ⅰ，Ⅱ | 适应带宽、数据包速率和流量大小分布的变化。 | 通用目标 |
| LD-Sketch[52] | Ⅰ | 记录部分流键并使用分布式架构。 | 重要流，重要变动，流大小 |
| UnivMon(O)[57] | - | 利用通用流式处理技术实现通用而准确的测量。 | 通用目标 |
| Defeat[71] | - | 将草图与子空间方法相结合。 | 异常检测 |
| Fast sketch[73] | Ⅳ | 将数据包聚合成少量的流。 | 重要变动 |
| TCM[74] | - | 使用一组图形草图来对流进行汇总和监测。 | 图表分析 |
| Bias-Aware Sketches[75] | Ⅱ | 在存在偏倚数据流的情况下提供错误保证。 | 重要流，重要变动，流大小 |
| ML Sketch[76] | Ⅱ | 在草图中引入机器学习理论以提高准确性。 | 通用目标 |
| Odd sketch[77] | - | 提出一种紧凑的二进制草图 | Jaccard相似 |

### 2.1.2 其他方法

基于草图的方法在细粒度网络测量中扮演着重要的角色。除了基于草图的方法之外，许多非草图算法使用不同的技术来实现相同的目标。

第一类是基于计数器的方法，例如Lossy Counting（参见第1节）。Space Saving[43]的目标是避免动态内存分配（参见第1节）并在Frequent[33]中添加频率估计。HashPipe[78]基于Space Saving算法来跟踪重要变化。Flajolet和Martin的概率计数[79]、线性计数[80]、LOGLOG[81]和HyperLogLog[82]用于测量流量的基数，特别是HyperLogLog在2017年扩展为HyperLogLog sketch[83]。Lall等人在[84]中提供了两种流算法来估计网络流量的熵。Zhang等人在[85]中提供了高分辨率的测量方法，以检测持续数十微秒的流量突发。Mitzenmacher等人在[86]和Ben-Basat等人在[44]中提出了处理分层重要变化问题的方法。CSE[87]旨在基于虚拟向量检测超级传播者和DDoS。快速哈希表[88]提高了数据包处理的性能。

由于标准布隆过滤器（见第1节）仅支持成员查询，因此提出了许多方法来改进它。光谱布隆过滤器（Spectral Bloom Filter，SBF）[89]扩展了标准布隆过滤器，支持多个集合，可以过滤掉重复次数小于阈值的项。可逆布隆查找表（Invertible Bloom Lookup Tables，IBLT）[90]适用于键值对，并且在时间内支持查找操作，其中是哈希函数的数量。IBLT能够准确计算流量监测中流数据包的频率。布隆树（Bloom Tree）[91]也适用于多个集合的成员测试，它使用完整的搜索树，在频率估计方面提供更高的准确性。计数商过滤器（Counting Quotient Filter，CQF）[50]提供了一种通用的近似成员查询方法，可以高效处理偏斜的输入数据，并支持合并和调整大小操作，因此适用于高效处理偏斜的流量流的成员测试。持久布隆过滤器（Persistent Bloom Filter，PBF）[92]可以对整个记录历史进行成员测试，在流量跟踪中非常有用。

一些测量方法使用经典的数学和统计理论来实现细粒度的测量。BeauCoup[62]是一个基于查询驱动的测量系统，专注于计数唯一任务（例如超级传播者[46]）。它基于优惠券收集问题，并且能够以比基于草图算法少4倍的内存访问次数实现相同的准确性。Zhang等人在[30]中开发了一种新颖的时空压缩感知框架，可以处理流量矩阵中的缺失值，并帮助更准确地测量网络。[31]使用了一种有损的数据结构：多分辨率计数器数组（Multi-Resolution Array of Counters，MRAC）来估计分布。

一些测量系统尝试通过利用网络中的不同实体（如端点主机、交换机、控制器）来实现更精细的测量或解决网络测量中尚未解决的问题。OmniMon[93]协调数据中心中的不同实体，以实现完全准确性和资源效率的目标。LightGuardian[56]是一个轻量级的带内遥测系统。当每个数据包经过交换机时，交换机会以概率的方式将称为sketchlet的草图列嵌入数据包头部。然后，端点主机收集这些数据包，并逐步从sketchlets中重建原始草图。cSamp[94]协调不同的路由器，实现全网范围的流量监测。NetSeer[95]是一个基于可编程交换机的流事件遥测系统，旨在发现和记录所有性能关键的数据平面事件（如数据包丢失、数据包暂停、拥塞）。[96]和iSTAMP[97]的测量模型基于OpenFlow[98]和TCAM构建了一个网络测量框架。NetSight[99]记录数据包的历史信息，用于进一步的流量分析。Pingmesh[100]旨在测量和分析网络延迟。Kandula等人在[101]和SNAP[102]中提出的工作通过收集套接字级别的日志来分析网络。FlowCover[103]、OpenSample[104]和OpenNetMon[105]在SDN中提供了流统计监测方案。Planck[106]使用端口镜像技术，以毫秒级的精度进行网络监测和交换机采样。EverFlow[107]使用“匹配和镜像”规则对数据包进行重排和过滤，从而减少开销。

**非草图方法的结论**：这些非草图方法利用各种理论和技术进行细粒度的测量。虽然这些算法不如基于草图的方法通用，但大多数可以针对特定的测量目标实现更高的性能（即更高的效率和准确率）。

## 2.2 控制层

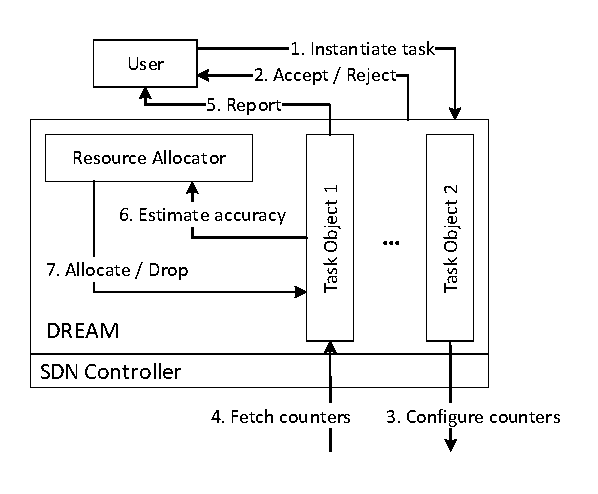


图 9 DREAM 系统框架图[35]

借助现有的新型测量数据结构和算法，我们可以在操作层面对各种目标进行精细化测量。然而，网络管理员要理解所有这些方法的特点并非易事。因此，我们需要在控制层面自动管理这些算法，以及调度多个测量任务。在本小节中，我们将探讨几个测量系统，着重于不同角度下的测量方法和测量任务的管理方面。

DREAM[35]专注于调度和分配当前可用资源，以满足不同任务需求和不同测量时间段的要求。作为一种基于TCAM的软件定义测量系统，它根据现有交换机上的TCAM硬件进行设计，因此可以立即在网络中部署使用。在DREAM中，用户可以向系统提交测量任务。提交的任务应包含四个参数：1）流过滤器，用于指定流量聚合；2）数据包头字段，用于定义任务事件；3）阈值，指定阈值容量；4）准确性限制，由用户指定。例如，用户可以提交一个重量级流量检测任务，流过滤器为<10/8, 12/8,\*,\*,\*>，阈值为1M，准确性为80%，并将源IP设置为数据包头字段。DREAM的工作流程详细如图9所示。步骤1，用户实例化测量任务并指定四个参数。步骤2，DREAM根据可用资源决定是否接受任务。步骤3，DREAM在一个或多个交换机上配置默认数量的计数器，并为接受的任务创建任务对象。步骤4，DREAM定期从交换机获取计数器，并将其传递给任务对象。步骤5，任务对象根据计数器计算测量结果，并基于计数器向用户生成报告。步骤6，任务对象通过准确性估算器测量当前任务的准确性，并将结果发送给资源分配器。步骤7，资源分配器确定为每个任务对象分配的TCAM计数器数量。任务对象使用分配的计数器来测量流量，并可以重新配置交换机（步骤3）。如果由于资源不足而丢弃任务，DREAM将删除任务对象并释放该任务的计数器。动态资源分配是DREAM的关键组成部分。它通过计算全局准确性和局部准确性来确定是否应该为任务分配更多资源。例如，如果任务的全局准确性超过指定限制，则无需分配更多资源，即使其中一个局部准确性低于限制。但如果全局准确性不满足要求，则局部准确性较低的交换机应为该任务分配更多计数器。在资源分配过程中，DREAM根据递减收益的原则，将计数器数量按照2的倍数进行调整。如果任务需要更多计数器，DREAM会分配两倍数量的计数器。如果任务较少且可以释放一些计数器，则DREAM会减半计数器数量。此外，操作员可以为每个任务指定丢弃优先级，使得低丢弃优先级的较差任务可以从高丢弃优先级的任务中获取资源。因此，具有高丢弃优先级的任务可能会被丢弃。对于大型交换机，DREAM几乎可以保持所有任务都能满足，无需拒绝或丢弃。对于小型交换机，它也可以保持较高的满意度，但可能会有更高的拒绝率。DREAM控制器能够处理大量任务，因为它具有高度的并行性。每个任务可以在一个核心上运行，每个交换机的分配器可以分别运行。控制循环的延迟包括保存增量规则、获取计数器、分配资源、生成报告、估算准确性和配置计数器等步骤，对于一个容量为512 TCAM的交换机，延迟不超过20毫秒。

OpenSketch[36]是一种软件定义网络测量架构，具有简单高效的数据平面和定制化的分析控制器。通过灵活的数据平面设计，OpenSketch(C)能够安装和管理多种不同的sketch算法，以确保准确性，并收集、分析测量数据，以满足应用层的需求。OpenSketch(C)在控制器上构建了一个测量库，可自动配置数据平面，使用不同的sketch。操作人员无需了解交换机的复杂实现，使测量编程更加简化。OpenSketch(C)的强大功能源于数据平面的创新设计。如图10所示，数据平面包含三个阶段：1) 哈希，2) 分类，3) 计数。哈希阶段选择要进行测量的数据包集，从而减少总测量数据的量。分类阶段可专注于通配规则指定的特定流量。计数阶段使用计数器表存储和累积流量统计信息，无需记录流键，以节省内存。因此，OpenSketch在使用哈希和分类模块时需要复杂的索引。OpenSketch(C)通过基于分类的索引维护存储计数器和特定流量之间的映射，或者通过基于哈希的索引从哈希值反向获取流量。通过巧妙地组合这三个阶段，数据平面能够支持多种不同的草图，以满足各种测量需求。OpenSketch(C)提供了一个草图库，其中包括草图管理器和资源分配器。草图管理器能够根据给定的测量任务要求自动选择合适的草图，并为实现最佳准确性而配置草图，并在不同草图之间分配资源，以实现可证明的内存-准确性权衡。草图管理器还能自动在数据平面中安装新的草图，以学习流量统计信息，以获得更好的配置。资源分配器能够自动将有限的资源分配给多个测量任务，无需考虑每个任务的具体实现细节，只需关注这些任务对操作员的相对重要性。借助资源分配器和草图管理器，OpenSketch(C)能够自动执行草图的管理，包括草图选择、草图大小设置等。OpenSketch(C)使用C++实现了七种草图，包括位图、哈希表、布隆过滤器、计数最小草图、可逆草图等，并且OpenSketch的数据平面已在NetFPGA上实现。实验结果显示，OpenSketch原型交换机在1GE端口上实现了全吞吐量且没有数据包丢失，即使对于64字节的数据包，每个测量组件的延迟也小于数据包的到达速率。然而，OpenSketch的数据平面稍微复杂，限制了数据包处理能力。在当前的10Gbps、40Gbps甚至100Gbps网络链路下，OpenSketch无法处理真实流量。

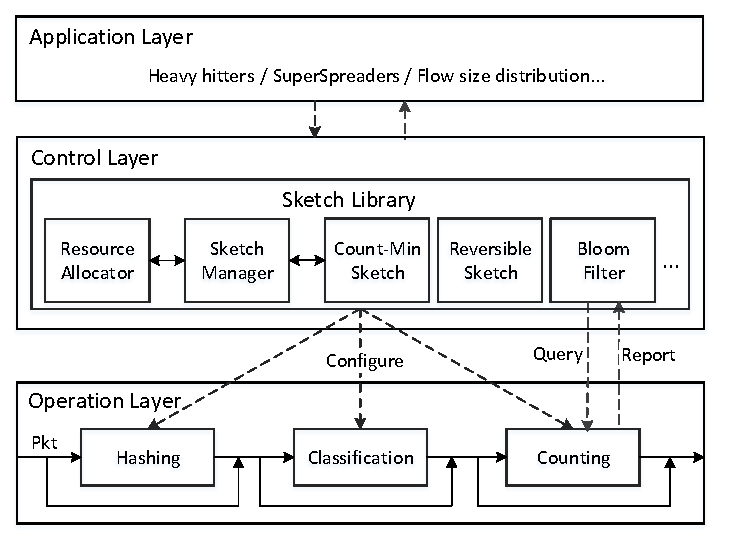


图 10 OpenSketch架构图[36]

SCREAM[55]是一种基于草图的资源分配测量系统，可视为DREAM[35]的改进版本，它克服了一些缺点：DREAM使用昂贵且耗电的TCAM内存，因此无法提供大量计数器以支持测量任务，而是依赖于基于前缀的汇总解决方案。相比之下，SCREAM利用廉价且高效的SRAM内存来使用草图总结流量。此外，SCREAM能够在不需要迭代重新配置的情况下捕获正确的流属性。它在多个交换机上实现了基于草图的任务，并通过准确性估计器实现了动态资源分配。SCREAM受到DREAM的启发，并有意地重新使用了动态资源分配器。SCREAM与DREAM主要在以下三个方面有所不同：1）SCREAM能够支持无法使用TCAM计数器支持的基于草图的任务。2）SCREAM可以为不同的交换机分配不同大小的草图，以适应它们可能观测到的不同流量量级，并将它们合并。3）SCREAM改进了DREAM的准确性估计器，以动态分配资源。SCREAM能够同时运行多个不同类型的任务实例，并将资源分配给每个交换机上的每个任务。每个任务定期从交换机查询计数器。根据流量观测结果，任务根据分配的资源更新草图参数，然后重新配置计数器。因此，SCREAM包含两个组件：任务和动态资源分配。SCREAM的总体架构如图11所示。控制器上的任务需要配置交换机上的草图计数器、获取计数器并准备报告。SCREAM针对不同的任务使用不同算法（如求和、最小值、最大值）来合并多个交换机上不同大小的草图。每个任务还包含一个准确性估计器，它使用马尔可夫不等式计算精度并估计准确性，然后定期共享结果和计数器。动态资源分配是SCREAM系统的核心，它使用任务的瞬时准确性作为迭代分配算法的反馈。如果估计的准确率小于指定的准确率界限，贫弱的任务将获得更多资源，这些资源从估计准确性远高于界限的富裕任务中分配出来。贫弱任务的资源分配顺序基于指定的优先级。如果无法分配给贫弱任务，则可能会丢弃该任务。在流量分布不断变化的情况下，相比于OpenSketch[36]，SCREAM能够通过动态资源分配支持更准确的任务。在单个交换机和多个交换机之间，SCREAM的满意度更高，拒绝率更低。准确性估计器的误差因任务类型而异，但随着交换机容量的增加，误差减小。实验表明，SCREAM的准确性估计误差在平均情况下控制在5%以内，所有交换机尺寸的召回率均保持在80%以上。

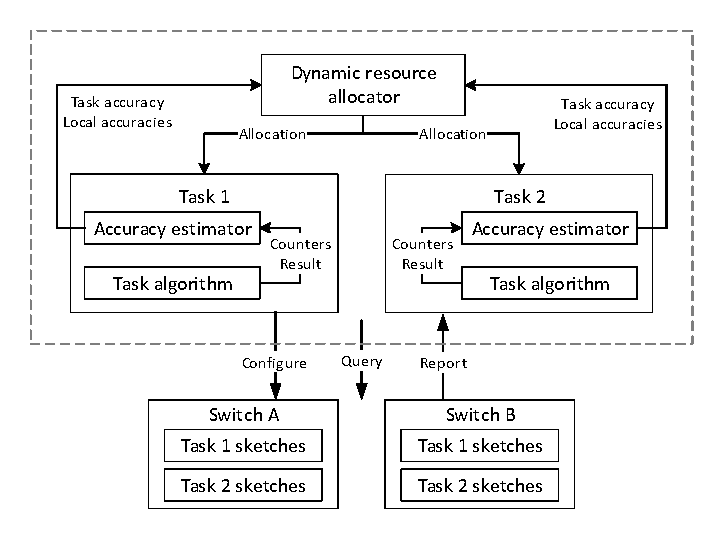


图 11 SCREAM框架图[55]

SketchLearn[38]旨在减轻网络管理员的负担，并支持对收集到的信息进行扩展查询。它通过对资源冲突的固有统计属性进行特征化，构建了一个多级草图，用于逐步推断和提取大流量，以确保剩余的流量较小。该迭代推断是自适应的，并且配置参数对最终结果的影响较小，从而减弱了配置与准确性之间的关联。SketchLearn的架构（见图12）由分布式数据平面和集中式控制平面组成。数据平面包含多个测量节点，每个节点都部署了一个多级草图，并处理传入的数据包。控制平面负责分析和解构多级草图，其中包括：1）大流量列表，用于标识大流量并记录估计的频率和相应的误差；2）剩余多级草图，用于存储剩余小流量的流量统计信息；3）位级计数器分布，每个分布模拟剩余多级草图中每个级别草图的计数器值分布。SketchLearn采用多级草图结构，其中包含多个小草图。每个小草图跟踪特定位数的流键（例如，5元组）的流量统计信息。假设流键的位数为，则多级草图包含个级别（从level-0到level-），每个级别对应一个具有行和列计数器的小草图。所有个草图使用相同的个哈希函数，其中第个哈希函数（）将流键映射到第行的第列（）。需要注意的是，level-0草图记录了所有数据包，而其他level-（）仅记录了流键的第位为1的数据包。假设表示第位为1的概率，记。多级草图具有一个重要特性，即如果没有大流量存在，将遵循以为均值的高斯分布。基于该特性，SketchLearn构建了一个冲突模型，其中包括大流量列表和剩余多级草图。该模型通过迭代地检查及其与的差异来推断和提取大流量，并将大流量从草图中删除并添加到大流量列表中，直到观察到的与高斯分布较好地吻合。SketchLearn支持通用的流量统计，并在各种测量任务中实现了高准确度，并且在不同的配置下保持稳定。SketchLearn的网络范围协调性能也表现出色。实验表明，随着测量点数量的增加，测量准确度显著提高。然而，多个草图使得SketchLearn在数据平面需要执行更多的哈希计算和计数器更新，可能导致性能下降。

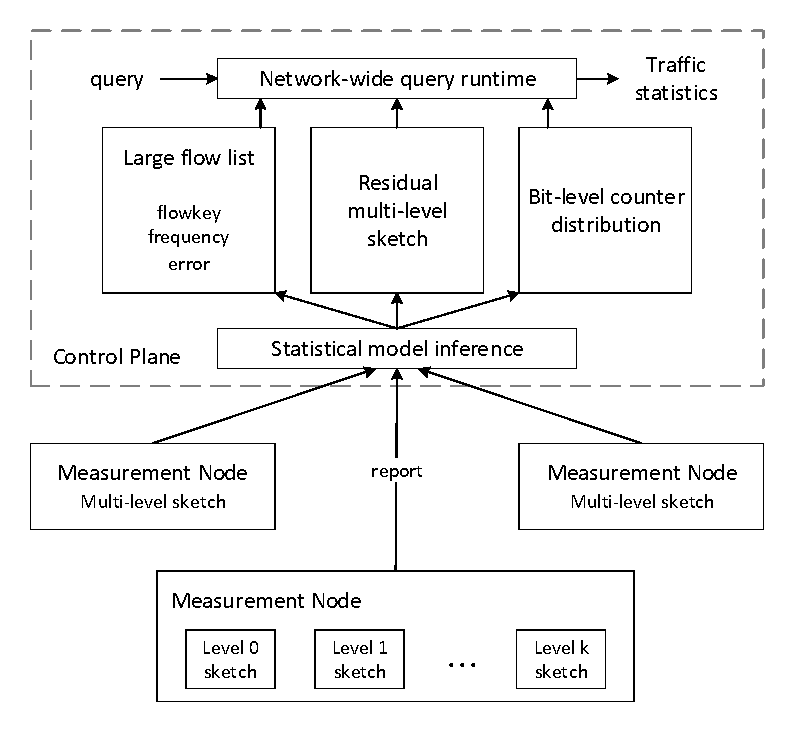


图 12 SCREAM框架图[55].

除了上述系统之外，还有许多其他在管理操作层的测量系统。UnivMon(C)[57]生成草图清单，以指定每个交换机的监控责任。当数据包到达交换机时，交换机使用清单确定要应用的草图操作集。UnivMon(C)从交换机收集草图信息，并为每个管理应用程序运行估算算法。自适应草图(Ada-Sketches)[108]提供时间自适应草图，根据时间动态调整草图大小，并重新分配有限的内存资源。弹性草图(C)(Elastic Sketch)[48]可以以高效和自适应的方式全局收集测量数据，是第一个提出草图压缩算法和合并策略的测量系统。SketchVisor(C)[53]旨在合并正常路径和快速路径的测量结果。它设计了一种恢复算法，使用矩阵插值和压缩感知来消除快速路径处理引起的额外误差。gSketch[109]使用网络特征（称为本地相似性）将整个图流的虚拟全局草图（例如CM草图）分区为一组本地化的草图，以实现更好的性能。

**控制层的结论**：在本节中，我们介绍了几种测量系统，它们提供了管理操作层现有方法的新机制。我们在表2中总结了它们的主要特点。然而，以上方法都不足以适当地管理复杂和多样化的测量方法。例如，它们大多需要用户指定要使用的测量方法。此外，它们没有考虑设备抽象层(DAL)，这意味着它们对于实际网络来说不够灵活和适应，实际网络可能包含不同类型的网络设备。因此，我们提出了控制层的四个要求：

* 解放网络管理员，减轻其选择和配置不同（且复杂）测量方法的负担。实现这一目标有两种方式。一种是根据具体的测量任务和底层数据平面自动选择适当的细粒度测量方法，另一种是使用简单的数据结构执行各种测量任务（如UnivMon[57]）。
* 根据流量特征的时变性，自适应地调整测量精度和资源占用之间的权衡。在大多数近似测量方法中，参数配置也是一项复杂的工作，因为测量精度与资源分配之间存在紧密关联。
* 支持并发执行多个测量任务。在诊断网络问题时，单一的测量指标可能不足以满足需求。网络管理员需要同时执行多个测量任务以确定根本原因。
* 全局合并测量数据并将分析结果报告给应用层，而不会占用过多的带宽和内存资源。测量任务可能在网络的不同位置运行。当控制层从多个交换机收集测量数据时，可能会造成带宽开销，并干扰正常流量，尤其是在高流量负载场景下。

表格 2 控制层测量系统的总结

|  |  |
| --- | --- |
| 测量系统 | 创新点 |
| DREAM [35] | 设计一个准确度估计器和一个资源分配器，以调度多个测量任务并管理操作层的测量算法。 |
| SCREAM [55] | 使用廉价和低功耗的SRAM内存（相对于TCAM）来管理基于草图的方法，并改进DREAM的准确度估计器，以动态分配资源。 |
| OpenSketch(C) [36] | 引入草图库，并提供一个草图管理器和资源分配器，以自动管理草图。 |
| SketchLearn [38] | 通过迭代学习草图的统计属性，自动解决资源冲突。 |
| UnivMon(C) [57] | 制定任务，然后将草图任务分配给网络元素，并提供大交换机抽象。 |
| Ada-Sketches [108] | 提供时适应型草图，可以动态调整草图大小并重新分配有限的内存资源。 |
| ElasticSketch(C) [48] | 通过引入草图压缩和合并策略来收集测量数据。 |
| SketchVisor(C) [53] | 使用矩阵插值和压缩感知来消除由于快速路径处理而产生的额外误差。 |
| gSketch [109] | 将虚拟全局草图划分为一组本地化草图，以实现更好的性能。 |

控制层在网络测量系统中比操作层和应用层更为重要，因为它扮演着管理者的角色，并处于中间位置。我们相信，在精细化网络测量的控制层仍有进一步改进的空间。除了管理操作层中的测量方法外，控制器还需要支持应用层上运行的多个应用程序。在下一小节中，我们将介绍几种测量接口，以不同的方式描述测量需求。

## 2.3 应用层

在面对网络管理问题时，网络管理员很难直接找到根本原因。网络管理员需要执行特定的测量任务，然后收集一些有用的信息来解决问题。在传统方式中，网络管理员需要凭借经验来决定要进行哪些测量，并手动执行测量任务，这都需要耗费大量时间。从软件定义的角度来看，这个过程可以抽象为由特定领域语言编程的测量应用程序。在本节中，我们将介绍近年来提出的几种应用层语言（或接口），它们可以用于执行网络查询或编写测量应用程序，从而使网络测量更加简便且可重复使用。

Frenetic[110]是一种专门针对OpenFlow网络编程的领域特定语言，它包含两个集成的子语言：一个是有限但高级和声明性的网络查询语言，另一个是通用的、功能性的和响应式的网络策略管理库。Frenetic嵌入在Python中，使编程更加便捷。Frenetic查询语言允许对一组数据包进行过滤，通过对标头字段进行分组来细分这些数据包集合，根据到达时间或标头字段值的变化将这些集合分割，限制返回值的数量，并对数据包进行聚合。接下来，我们来介绍几个主要的语法元素。如图13所示，子句使用方法对结果进行聚合，其中可以是数据包、计数或字节数。子句对结果进行过滤，并保留满足过滤模式的数据包。子句根据标头字段到将数据包集合分成子集。子句也将数据包分成子集，不同之处在于它在给定字段的值发生变化时生成一个新的子集。例如，如果查询按源IP地址进行拆分，具有源IP地址A、B、A的数据包流将被拆分为三个子集，因为它们的IP地址与前一个不同。如果流序列是A、A、B，则只生成两个子集。子句将数据包分组，同一秒的时间窗口内到达的数据包将被分组。子句限制每个子集中的数据包数量。查询的结果是一个事件流，表示一系列的值。该库通过功能性和响应式编程来帮助管理数据包转发策略。Frenetic程序的一个基本操作是构建用于安装在交换机上的数据包转发规则。因此，Frenetic程序可以使用该库创建网络策略，并控制在网络中安装策略。Frenetic的查询语言定义了需要监视的数据包类型。该语言允许测量具有特定字段值的数据包，并在测量特定任务时仅关注流量的必要部分。这个查询语言提供了一个通用的过滤器，可以显著减少数据包规模，然后再由草图或计数器元组进行处理。

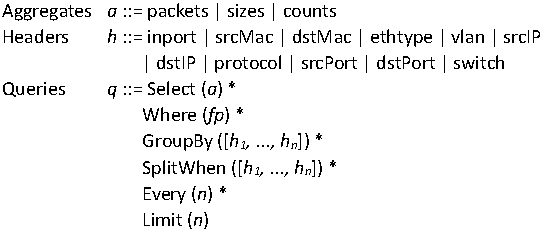


图 13 Frenetic查询的语法[110]

Path Query[111]是一种测量语言，不仅用于流量测量，还可用于整个网络监控。Narayana等人在[112]中开发了一种路径查询语言，其中运算符通过对数据包的位置和标头内容进行布尔条件的正则表达式匹配。网络中的交换机被编程为在数据包流经时记录路径信息。查询将被编译成分布在各个交换机上的确定有限自动机（DFA）。DFA的状态存储在每个数据包中，并在数据包经过网络时进行更新。交换机在接收数据包时读取DFA的当前状态，然后检查查询条件，并将新的状态写入数据包中。路径查询用于识别具有特定标头值并经过特定位置的数据包集合。布尔谓词是路径查询的基本组成部分，可在单个位置匹配数据包，并可在标准标头字段（如源IP）上进行匹配。谓词可以与拓扑相关联。用于匹配从特定入口接口进入网络的所有数据包，而用于匹配从特定出口接口退出网络的所有数据包。由于数据包头部重写或数据包丢失的原因，与给定谓词匹配的数据包集合在交换机入口和出口处可能不同，因此引入了来细化谓词的含义。进入交换机时测试数据包，在离开时测试数据包。数据包进入时测试一个谓词，离开时测试另一个谓词。还支持分组操作，例如，在交换机入口收集与匹配的数据包，并根据标头值将它们分成不同的组。和的功能类似。路径查询可以通过使用正则组合器来描述原子的组合方式：表示两者都满足，表示两者有一个满足，表示不满足，表示连接，查询表示满足当前交换机上的路径，下一跳交换机满足；表示重复，表示满足的路径可以重复出现零次或多次。语法摘要如图14所示。匹配查询条件的数据包可以进行计数、发送到特定端口或SDN控制器等操作。运算符还可以指定在路径上的哪个位置捕获满足查询条件的数据包，可以是上游、下游或中间位置。

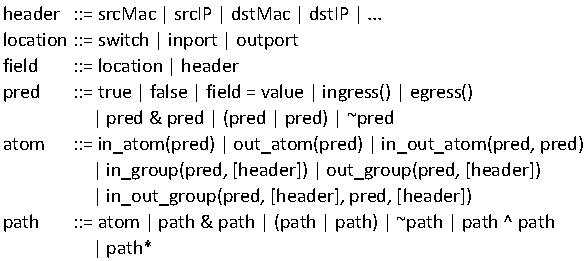


图 14 Path Query 的语法[111]

Marple[54]提供了一种扩展了Path Query的语言，可以表达各种性能监测的用例。借助专用硬件，Marple语言可以获得更详细的交换机信息，如队列长度、进出交换机的数据包时间戳等。Marple语言的设计目标不仅仅是简单的流量测量，而是旨在对其监测的流进行性能分析。Marple提供了网络流性能信息的抽象表示。这些流包含元组，记录了性能元数据，如数据包进入队列时的队列长度和时间戳。然后，这些流被记录为，并作为基本输入流。Marple为每个队列中的每个数据包提供了一个元组，包括七个字段：。其中，和表示观察到数据包的交换机和队列。可以记录常规数据包标头，而唯一确定一个数据包。和分别表示该数据包的入队和出队时间戳，而表示数据包入队时的队列长度。基于这些元数据，Marple提供了四个功能构造：、、和，具体如表3所示。的形式为，其中是某个，谓词可以限制用户的关注范围，仅选择满足谓词条件的数据包。的形式为，可以计算字段以表达感兴趣的新指标。例如，可以将数据包时间戳舍入到新的字段。的形式为，其中输入流将按进行分组，而聚合函数将对每个组进行操作。例如，可以统计属于同一个5元组流的数据包数量。的形式为，可以合并字段。操作的输出是一个单一流，其中的元组是两个输入流中所有字段的组合。Marple语言允许用户根据和等过滤长时间等待的数据包，或者根据特定队列进行过滤。同时，它还提供了常规的标头字段过滤规则。因此，Marple语言主要用于监测网络性能，这在流量测量中具有重要的应用价值。

表格 3 Marple语言功能构造的总结

|  |  |
| --- | --- |
| 功能构造 | 描述 |
|  | 在中输出满足谓词的元组 |
|  | 在字段上评估，发送有新字段的元组 |
|  | 根据划分，在划分的组上操作函数 |
|  | 将和两个元组合并 |

Sonata[111]是一个允许运营商使用熟悉的数据流操作符（例如 map、filter、reduce）来表达查询的系统。它指出，Maple[54]和OpenSketch[36]的局限在于它们仅在数据平面中执行查询，内存不足且可编程性较差。Sonata允许运营商将每个数据包视为元组，并将查询表示为对元组流的操作。它在交换机和流处理器之间分配工作负载，并迭代地对数据面（交换机）和处理器进行配置，以更细粒度的方式检查流量。如图15所示，Sonata将用户查询编译为一组安装在交换机中的规则，并在流处理器上安装处理管道。数据包被视为元组，每个元组是一组包括源IP、源MAC、源端口、目标IP、目标MAC、目标端口等字段值的集合。运营商可以指定特定操作是在交换机中执行还是在流处理器上执行。例如，运营商可以在交换机上应用过滤器A，在处理器上应用另一个过滤器B。Sonata还允许运营商表达查询的细化逻辑。进行中的查询结果可以驱动对现有查询的细化，并迭代地进行查询的优化。

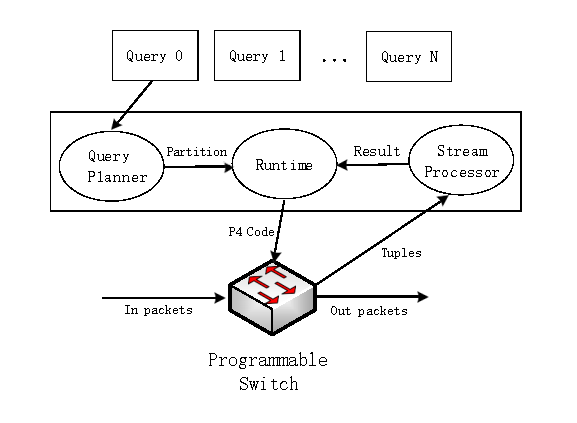


图 15 Sonata的概况[111]

除了上述接口，OpenSketch(A)[36]提供了配置和查询API，使用户能够进行网络测量，而无需考虑具体的草图实现。ProgME[113]提出了流集合组合语言（FCL），用于基于流集合收集流量统计信息。TPP[114]使用编程接口，并将微型程序嵌入数据包中以获取网络状态。Stroboscope[115]引入了一种类似于SQL的语言，通过激活和停用目标前缀的流量镜像来提取流量样本集。

**应用层的结论**：所有上述的测量语言或接口都能以不同的方式描述测量任务。我们在表4中对应用层进行了总结。然而，这些语言对于各种测量任务来说都不够通用。更重要的是，在处理类似于[110]和[58]这样的有状态查询操作时，它们会引入显著的性能损失。因此，我们提出了测量语言或接口的四个要求：

* 提供通用接口，以定义不同类型的测量任务：期望能够支持各种管理操作，因此需要一个通用接口框架来定义所有类型的测量目标（例如，重要流量、DDoS攻击）。
* 能够编译为一组细粒度的测量方法：这一点非常重要，因为我们需要利用现有的细粒度方法来进行网络测量。
* 在运行之前明确性能开销：同样非常重要，因为某些测量任务不可避免地会引入显著的性能损失[54]。在网络管理员执行测量任务之前需要提前向其告知。
* 近实时的响应时间：响应时间包括两个方面。首先是由于转换和安装测量方法而引入的时间，其次是测量数据能够被收集和分析的时间。这两个方面都需要高效。

测量语言或接口的设计是一个有趣但具有挑战性的课题。对于细粒度测量系统，我们需要更加通用和灵活的语言或接口。

表格 4 测量语言的对比

|  |  |  |
| --- | --- | --- |
| 语言 | 用途 | 要求 |
| Frenetic[110] | 使用特定字段值对数据包进行过滤。 | Python |
| Path Query [112] | 追踪包路径 | 支持路径查询的可编程交换机 |
| Marple [54] | 监控网络性能 | 具有可编程键值存储的交换机。 |
| Sonata [111] | 监控网络性能 | 可编程交换机，流处理器 |
| ProgME [113] | 将数据包分组到任意一组流 | 一个基于二进制决策图的数据结构和基于流集合的查询回答引擎 |
| TPP [114] | 收集包历史 | 结合TPP的NetFPGA |
| Stroboscope [115] | 镜像和分析流量 | 支持镜像的交换机 |

# 3. 结论

在本文中，我们对最近的细粒度测量方法和系统进行了全面调研。我们从软件定义的角度讨论了这些方法和系统。具体而言，我们将现有的工作分为三个层次，并讨论了每个层次的当前限制或未来需求。

对于细粒度网络测量的未来发展方向，我们认为控制层有很大的改进空间。与其他网络操作相比，网络测量的执行过程仍然相对原始，特别是在以细粒度和动态方式测量高速网络时。细粒度测量缺乏强大的控制平面是主要原因。测量接口可以被视为后续的产物，因为它由相应的控制器进行编译和执行。最后，在操作层上，需要更加高效和准确的测量算法，以满足既有和新的测量需求。

# 参考文献

1. marketsandmarkets, “Network analytics market by type, by end user - Global forecast to 2020,”[EB/OL]. (2019). <https://www.marketsandmarkets.com/Market-Reports/network-analytics-market-1244.html>
2. Y. Khalidi, “Azure networking fall 2018 update,”.[EB/OL] (2018). <https://azure.microsoft.com/en-us/blog/azure-networking-fall-2018-update>
3. I. Envid, “Introducing VPC flow logs network transparency in near real-time,”.[EB/OL] (2018). <https://cloud.google.com/blog/products/gcp/introducing-vpc-flow-logs-networktransparency-in-near-real-time>
4. Claise B. Cisco systems netflow services export version 9[R]. 2004.
5. Wang M, Li B, Li Z. sFlow: Towards resource-efficient and agile service federation in service overlay networks[C]//24th International Conference on Distributed Computing Systems, 2004. Proceedings. IEEE, 2004: 628-635.
6. Manku G S, Motwani R. Approximate frequency counts over data streams[C]//VLDB'02: Proceedings of the 28th International Conference on Very Large Databases. Morgan Kaufmann, 2002: 346-357.
7. Estan C, Varghese G. New directions in traffic measurement and accounting[C]//Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications. 2002: 323-336.
8. Zhang Y. An adaptive flow counting method for anomaly detection in SDN[C]//Proceedings of the ninth ACM conference on Emerging networking experiments and technologies. 2013: 25-30.
9. Canini M, Fay D, Miller D J, et al. Per flow packet sampling for high-speed network monitoring[C]//2009 First International Communication Systems and Networks and Workshops. IEEE, 2009: 1-10.
10. Stutzbach D, Rejaie R, Duffield N, et al. On unbiased sampling for unstructured peer-to-peer networks[C]//Proceedings of the 6th ACM SIGCOMM conference on Internet measurement. 2006: 27-40.
11. Mendiola A, Astorga J, Jacob E, et al. A survey on the contributions of software-defined networking to traffic engineering[J]. IEEE Communications Surveys & Tutorials, 2016, 19(2): 918-953.
12. Agarwal S, Kodialam M, Lakshman T V. Traffic engineering in software defined networks[C]//2013 Proceedings IEEE INFOCOM. IEEE, 2013: 2211-2219.
13. Shafiee M, Ghaderi J. A simple congestion-aware algorithm for load balancing in datacenter networks[J]. IEEE/ACM Transactions on Networking, 2017, 25(6): 3670-3682.
14. Cormode G, Muthukrishnan S. An improved data stream summary: the count-min sketch and its applications[J]. Journal of Algorithms, 2005, 55(1): 58-75.
15. Cormode G, Muthukrishnan S. What's new: Finding significant differences in network data streams[J]. IEEE/ACM Transactions on Networking, 2005, 13(6): 1219-1232.
16. Schweller R, Li Z, Chen Y, et al. Reversible sketches: enabling monitoring and analysis over high-speed data streams[J]. IEEE/ACM Transactions on Networking, 2007, 15(5): 1059-1072.
17. Roy P, Khan A, Alonso G. Augmented sketch: Faster and more accurate stream processing[C]//Proceedings of the 2016 International Conference on Management of Data. 2016: 1449-1463.
18. Zhou Y, Yang T, Jiang J, et al. Cold filter: A meta-framework for faster and more accurate stream processing[C]//Proceedings of the 2018 International Conference on Management of Data. 2018: 741-756.
19. Ting D. Data sketches for disaggregated subset sum and frequent item estimation[C]//Proceedings of the 2018 International Conference on Management of Data. 2018: 1129-1140.
20. P4 language. [EB/OL]. <https://p4.org/>
21. Pfaff B, Pettit J, Koponen T, et al. The design and implementation of open vSwitch[C]//12th USENIX symposium on networked systems design and implementation (NSDI 15). 2015: 117-130.
22. Bess: Berkeley extensible software switch. [EB/OL] <https://github.com/NetSys/bess>
23. Tofino 2. [Online]. [EB/OL] <https://www.barefootnetworks.com/products/brief-tofino-2/>
24. Scrutinizer. [EB/OL] <https://www.plixer.com/products/scrutinizer/>
25. Netflow analyzer PRTG. [EB/OL] <https://www.paessler.com/netflow_monitoring>
26. Hofstede R, Čeleda P, Trammell B, et al. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix[J]. IEEE Communications Surveys & Tutorials, 2014, 16(4): 2037-2064.
27. Estan C, Keys K, Moore D, et al. Building a better NetFlow[J]. ACM SIGCOMM Computer Communication Review, 2004, 34(4): 245-256.
28. Duffield N, Lund C, Thorup M. Estimating flow distributions from sampled flow statistics[C]//Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications. 2003: 325-336.
29. Hohn N, Veitch D. Inverting sampled traffic[C]//Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement. 2003: 222-233.
30. Zhang Y, Roughan M, Willinger W, et al. Spatio-temporal compressive sensing and internet traffic matrices[C]//Proceedings of the ACM SIGCOMM 2009 conference on Data communication. 2009: 267-278.
31. Kumar A, Sung M, Xu J, et al. Data streaming algorithms for efficient and accurate estimation of flow size distribution[J]. ACM SIGMETRICS Performance Evaluation Review, 2004, 32(1): 177-188.
32. Charikar M, Chen K, Farach-Colton M. Finding frequent items in data streams[C]//International Colloquium on Automata, Languages, and Programming. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002: 693-703.
33. Demaine E D, López-Ortiz A, Munro J I. Frequency estimation of internet packet streams with limited space[C]//European Symposium on Algorithms. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002: 348-360.
34. Manku G S, Motwani R. Approximate frequency counts over data streams[C]//VLDB'02: Proceedings of the 28th International Conference on Very Large Databases. Morgan Kaufmann, 2002: 346-357.
35. Moshref M, Yu M, Govindan R, et al. Dream: dynamic resource allocation for software-defined measurement[C]//Proceedings of the 2014 ACM conference on SIGCOMM. 2014: 419-430.
36. Yu M, Jose L, Miao R. Software Deﬁned Trafﬁc Measurement with OpenSketch[C]//10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13). 2013: 29-42.
37. Shan J, Fu Y, Ni G, et al. Fast counting the cardinality of flows for big traffic over sliding windows[J]. Frontiers of Computer Science, 2017, 11: 119-129.
38. Huang Q, Lee P P C, Bao Y. Sketchlearn: Relieving user burdens in approximate measurement with automated statistical inference[C]//Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. 2018: 576-590.
39. D’Alconzo A, Drago I, Morichetta A, et al. A survey on big data for network traffic monitoring and analysis[J]. IEEE Transactions on Network and Service Management, 2019, 16(3): 800-813.
40. Wang L, Jones R. Big data analytics for network intrusion detection: A survey[J]. International Journal of Networks and communications, 2017, 7(1): 24-31.
41. So-In C. A survey of network traffic monitoring and analysis tools[J]. Cse 576m computer system analysis project, Washington University in St. Louis, 2009.
42. Azzana Y, Chabchoub Y, Fricker C, et al. Adaptive algorithms for identifying large flows in IP traffic[J]. arXiv preprint arXiv:0901.4846, 2009.
43. Metwally A, Agrawal D, El Abbadi A. Efficient computation of frequent and top-k elements in data streams[C]//International conference on database theory. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005: 398-412.
44. Ben Basat R, Einziger G, Friedman R, et al. Constant time updates in hierarchical heavy hitters[C]//Proceedings of the Conference of the ACM Special Interest Group on Data Communication. 2017: 127-140.
45. Cormode G, Muthukrishnan S. What's new: Finding significant differences in network data streams[J]. IEEE/ACM Transactions on Networking, 2005, 13(6): 1219-1232.
46. Kamiyama N, Mori T, Kawahara R. Simple and adaptive identification of superspreaders by flow sampling[C]//IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications. IEEE, 2007: 2481-2485.
47. Xu Y, Liu Y. DDoS attack detection under SDN context[C]//IEEE INFOCOM 2016-the 35th annual IEEE international conference on computer communications. IEEE, 2016: 1-9.
48. Yang T, Jiang J, Liu P, et al. Elastic sketch: Adaptive and fast network-wide measurements[C]//Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. 2018: 561-575.
49. Bloom B H. Space/time trade-offs in hash coding with allowable errors[J]. Communications of the ACM, 1970, 13(7): 422-426.
50. Pandey P, Bender M A, Johnson R, et al. A general-purpose counting filter: Making every bit count[C]//Proceedings of the 2017 ACM international conference on Management of Data. 2017: 775-787.
51. Cormode G, Hadjieleftheriou M. Finding frequent items in data streams[J]. Proceedings of the VLDB Endowment, 2008, 1(2): 1530-1541.
52. Huang Q, Lee P P C. A hybrid local and distributed sketching design for accurate and scalable heavy key detection in network data streams[J]. Computer Networks, 2015, 91: 298-315.
53. Huang Q, Jin X, Lee P P C, et al. Sketchvisor: Robust network measurement for software packet processing[C]//Proceedings of the Conference of the ACM Special Interest Group on Data Communication. 2017: 113-126.
54. Narayana S, Sivaraman A, Nathan V, et al. Language-directed hardware design for network performance monitoring[C]//Proceedings of the Conference of the ACM Special Interest Group on Data Communication. 2017: 85-98.
55. Moshref M, Yu M, Govindan R, et al. Scream: Sketch resource allocation for software-defined measurement[C]//Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies. 2015: 1-13.
56. Zhao Y, Yang K, Liu Z, et al. {LightGuardian}: A {full-visibility}, lightweight, in-band telemetry system using sketchlets[C]//18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21). 2021: 991-1010.
57. Liu Z, Manousis A, Vorsanger G, et al. One sketch to rule them all: Rethinking network flow monitoring with univmon[C]//Proceedings of the 2016 ACM SIGCOMM Conference. 2016: 101-114.
58. Gupta A, Harrison R, Canini M, et al. Sonata: Query-driven streaming network telemetry[C]//Proceedings of the 2018 conference of the ACM special interest group on data communication. 2018: 357-371.
59. Lu Y, Montanari A, Prabhakar B, et al. Counter braids: a novel counter architecture for per-flow measurement[J]. ACM SIGMETRICS Performance Evaluation Review, 2008, 36(1): 121-132.
60. Yang T, Zhou Y, Jin H, et al. Pyramid sketch: A sketch framework for frequency estimation of data streams[J]. Proceedings of the VLDB Endowment, 2017, 10(11): 1442-1453.
61. Liu Z, Ben-Basat R, Einziger G, et al. Nitrosketch: Robust and general sketch-based monitoring in software switches[M]//Proceedings of the ACM Special Interest Group on Data Communication. 2019: 334-350.
62. Chen X, Landau-Feibish S, Braverman M, et al. Beaucoup: Answering many network traffic queries, one memory update at a time[C]//Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. 2020: 226-239.
63. Cormode G, Muthukrishnan S. An improved data stream summary: the count-min sketch and its applications[J]. Journal of Algorithms, 2005, 55(1): 58-75.
64. Tang L, Huang Q, Lee P P C. Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams[C]//IEEE INFOCOM 2019-IEEE Conference on Computer Communications. IEEE, 2019: 2026-2034.
65. Li Y, Miao R, Kim C, et al. FlowRadar: A Better NetFlow for Data Centers[C]//13th USENIX symposium on networked systems design and implementation (NSDI 16). 2016: 311-324.
66. Tang L, Huang Q, Lee P P C. SpreadSketch: Toward invertible and network-wide detection of superspreaders[C]//IEEE INFOCOM 2020-IEEE Conference on Computer Communications. IEEE, 2020: 1608-1617.
67. Bu T, Cao J, Chen A, et al. Sequential hashing: A flexible approach for unveiling significant patterns in high speed networks[J]. Computer Networks, 2010, 54(18): 3309-3326.
68. Huang Q, Sheng S, Chen X, et al. Toward {Nearly-Zero-Error} sketching via compressive sensing[C]//18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21). 2021: 1027-1044.
69. Goyal A, Daumé H. Lossy conservative update (LCU) sketch: Succinct approximate count storage[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2011, 25(1): 878-883.
70. Liu Z, Vorsanger G, Braverman V, et al. Enabling a" risc" approach for software-defined monitoring using universal streaming[C]//Proceedings of the 14th ACM Workshop on Hot Topics in Networks. 2015: 1-7.
71. Li X, Bian F, Crovella M, et al. Detection and identification of network anomalies using sketch subspaces[C]//Proceedings of the 6th ACM SIGCOMM conference on Internet measurement. 2006: 147-152.
72. Lakhina A, Crovella M, Diot C. Mining anomalies using traffic feature distributions[J]. ACM SIGCOMM computer communication review, 2005, 35(4): 217-228.
73. Liu Y, Chen W, Guan Y. A fast sketch for aggregate queries over high-speed network traffic[C]//2012 Proceedings IEEE INFOCOM. IEEE, 2012: 2741-2745.
74. Tang N, Chen Q, Mitra P. Graph stream summarization: From big bang to big crunch[C]//Proceedings of the 2016 International Conference on Management of Data. 2016: 1481-1496.
75. J. Chen and Q. Zhang, Bias-aware sketches[J]. Proc. VLDB Endowment, 2017, 10(9): 961–972.
76. Yang T, Wang L, Shen Y, et al. Empowering sketches with machine learning for network measurements[C]//Proceedings of the 2018 Workshop on Network Meets AI & ML. 2018: 15-20.
77. Mitzenmacher M, Pagh R, Pham N. Efficient estimation for high similarities using odd sketches[C]//Proceedings of the 23rd international conference on World wide web. 2014: 109-118.
78. Mitzenmacher M, Pagh R, Pham N. Efficient estimation for high similarities using odd sketches[C]//Proceedings of the 23rd international conference on World wide web. 2014: 109-118.
79. Flajolet P, Martin G N. Probabilistic counting algorithms for data base applications[J]. Journal of computer and system sciences, 1985, 31(2): 182-209.
80. Whang K Y, Vander-Zanden B T, Taylor H M. A linear-time probabilistic counting algorithm for database applications[J]. ACM Transactions on Database Systems (TODS), 1990, 15(2): 208-229.
81. Durand M, Flajolet P. Loglog counting of large cardinalities[C]//Algorithms-ESA 2003: 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003. Proceedings 11. Springer Berlin Heidelberg, 2003: 605-617.
82. Flajolet P, Fusy É, Gandouet O, et al. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm[J]. Discrete mathematics & theoretical computer science, 2007 (Proceedings).
83. Ertl O. New cardinality estimation algorithms for HyperLogLog sketches[J]. arXiv preprint arXiv:1702.01284, 2017.
84. Lall A, Sekar V, Ogihara M, et al. Data streaming algorithms for estimating entropy of network traffic[J]. ACM SIGMETRICS Performance Evaluation Review, 2006, 34(1): 145-156.
85. Zhang Q, Liu V, Zeng H, et al. High-resolution measurement of data center microbursts[C]//Proceedings of the 2017 Internet Measurement Conference. 2017: 78-85.
86. Mitzenmacher M, Steinke T, Thaler J. Hierarchical heavy hitters with the space saving algorithm[C]//2012 Proceedings of the Fourteenth Workshop on Algorithm Engineering and Experiments (ALENEX). Society for Industrial and Applied Mathematics, 2012: 160-174.
87. Mitzenmacher M, Steinke T, Thaler J. Hierarchical heavy hitters with the space saving algorithm[C]//2012 Proceedings of the Fourteenth Workshop on Algorithm Engineering and Experiments (ALENEX). Society for Industrial and Applied Mathematics, 2012: 160-174.
88. Song H, Dharmapurikar S, Turner J, et al. Fast hash table lookup using extended bloom filter: an aid to network processing[J]. ACM SIGCOMM Computer Communication Review, 2005, 35(4): 181-192.
89. Cohen S, Matias Y. Spectral bloom filters[C]//Proceedings of the 2003 ACM SIGMOD international conference on Management of data. 2003: 241-252.
90. Goodrich M T, Mitzenmacher M. Invertible bloom lookup tables[C]//2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton). IEEE, 2011: 792-799.
91. Yoon M K, Son J W, Shin S H. Bloom tree: A search tree based on bloom filters for multiple-set membership testing[C]//IEEE INFOCOM 2014-IEEE Conference on Computer Communications. IEEE, 2014: 1429-1437.
92. Peng Y, Guo J, Li F, et al. Persistent bloom filter: Membership testing for the entire history[C]//Proceedings of the 2018 International Conference on Management of Data. 2018: 1037-1052.
93. Huang Q, Sun H, Lee P P C, et al. Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy[C]//Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. 2020: 404-421.
94. Sekar V, Reiter M K, Willinger W, et al. cSamp: A system for network-wide flow monitoring[J]. 2008.
95. Zhou Y, Sun C, Liu H H, et al. Flow event telemetry on programmable data plane[C]//Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. 2020: 76-89.
96. Jose L, Yu M. Online measurement of large traffic aggregates on commodity switches[C]//Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 11). 2011.
97. Malboubi M, Wang L, Chuah C N, et al. Intelligent SDN based traffic (de) aggregation and measurement paradigm (iSTAMP)[C]//IEEE INFOCOM 2014-IEEE Conference on Computer Communications. IEEE, 2014: 934-942.
98. Malboubi M, Wang L, Chuah C N, et al. Intelligent SDN based traffic (de) aggregation and measurement paradigm (iSTAMP)[C]//IEEE INFOCOM 2014-IEEE Conference on Computer Communications. IEEE, 2014: 934-942.
99. Handigol N, Heller B, Jeyakumar V, et al. I know what your packet did last hop: Using packet histories to troubleshoot networks[C]//11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). 2014: 71-85.
100. Guo C, Yuan L, Xiang D, et al. Pingmesh: A large-scale system for data center network latency measurement and analysis[C]//Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. 2015: 139-152.
101. Kandula S, Sengupta S, Greenberg A, et al. The nature of data center traffic: measurements & analysis[C]//Proceedings of the 9th ACM SIGCOMM conference on Internet measurement. 2009: 202-208.
102. Yu M, Greenberg A, Maltz D, et al. Profiling network performance for multi-tier data center applications[C]//8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11). 2011.
103. Su Z, Wang T, Xia Y, et al. FlowCover: Low-cost flow monitoring scheme in software defined networks[C]//2014 IEEE Global Communications Conference. IEEE, 2014: 1956-1961.
104. Suh J, Kwon T T, Dixon C, et al. Opensample: A low-latency, sampling-based measurement platform for commodity sdn[C]//2014 IEEE 34th International Conference on Distributed Computing Systems. IEEE, 2014: 228-237.
105. Van Adrichem N L M, Doerr C, Kuipers F A. Opennetmon: Network monitoring in openflow software-defined networks[C]//2014 IEEE Network Operations and Management Symposium (NOMS). IEEE, 2014: 1-8.
106. Rasley J, Stephens B, Dixon C, et al. Planck: Millisecond-scale monitoring and control for commodity networks[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(4): 407-418.
107. Zhu Y, Kang N, Cao J, et al. Packet-level telemetry in large datacenter networks[C]//Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. 2015: 479-491.
108. Shrivastava A, Konig A C, Bilenko M. Time adaptive sketches (ada-sketches) for summarizing data streams[C]//Proceedings of the 2016 International Conference on Management of Data. 2016: 1417-1432.
109. Zhao P, Aggarwal C C, Wang M. gsketch: On query estimation in graph streams[J]. arXiv preprint arXiv:1111.7167, 2011.
110. Foster N, Harrison R, Freedman M J, et al. Frenetic: A network programming language[J]. ACM Sigplan Notices, 2011, 46(9): 279-291.
111. Gupta A, Birkner R, Canini M, et al. Network monitoring as a streaming analytics problem[C]//Proceedings of the 15th ACM workshop on hot topics in networks. 2016: 106-112.
112. Gupta A, Birkner R, Canini M, et al. Network monitoring as a streaming analytics problem[C]//Proceedings of the 15th ACM workshop on hot topics in networks. 2016: 106-112.
113. Gupta A, Birkner R, Canini M, et al. Network monitoring as a streaming analytics problem[C]//Proceedings of the 15th ACM workshop on hot topics in networks. 2016: 106-112.
114. Jeyakumar V, Alizadeh M, Geng Y, et al. Millions of little minions: Using packets for low latency network programming and visibility[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(4): 3-14.
115. Tilmans O, Bühler T, Poese I, et al. Stroboscope: Declarative network monitoring on a budget[C]//15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18). 2018: 467-482.