一种资源节约的快速一维网络流量聚合方法

江洁欣[1,2]，程光[1,3]

（1.东南大学 计算机科学与工程学院，南京 210096；2.江苏省计算机网络重点实验室， 南京 210096；
3. 东南大学计算机网络和信息集成教育部重点实验室）

摘要 网络流量是一种具有多维属性的数据，流量聚合的目的是掌握链路上的主要流量模式。一维聚合是流量聚合的最简单情况，同时也是多维聚合的基础。文章通过分析流量一维聚合的主要时间和空间消耗，提出了一种基于层次域的流量一维聚合算法，通过采用自顶向下的聚合策略避免在层次树中创建非重聚合点，从而减少算法总体运行时间并大大降低了内存占用。

关键词 网络测量，流量聚合，流信息挖掘

1 研究背景

网络管理员必须随时掌握链路上的流量信息以便及时对网络进行管理，然后针对性的优化、部署和调度各种资源。在采集到流量数据后，需要对数据进行分析整理，提取有价值的摘要信息报告管理员。

采集到的原始流量是不可用的，因为流量信息的详细程度和管理员的理解能力是一对矛盾；详细的报文级信息虽然能无损的反映网络状况，但对管理员来说是完全不可用的。从实用角度出发，流量报告需要提供尽可能详细、可用的信息，使管理员只通过少量的摘要信息就能够掌握全局概况。另一方面，由于高速链路流量巨大，采集到的原始流量需要按照一些标准进行聚合，提取其中的“主要”流量模式，它们的流量超过总流量一定百分比的流量。

流量聚合用于解决这个问题。网络流量具有多维属性，为了全面了解流量构成和主要流量模式，需要对原始流量进行一维和多维的聚合，将聚合后超过总流量百分比的流量模式提取出来作为对流量构成的总体概括。一维聚合是指在一个维度上对流量进行聚合，是流量聚合的最简单情况，也是多维流量聚合算法的基础。

2 相关工作

C.Estan 在[1]中提出一种基于层次域的一维聚合算法，将五元组流定义的五个维度上的取值空间形成层次域，把流量按照层次之间的包含关系进行聚合，离线计算重聚合点。Wang 在[2]中对 Estan 的方法进行改进，认为算法的主要消耗发生在潜在的排序操作上，提出了一种自顶向下的聚合方法，以流记录的反复查找代替排序操作并通过一些优化手段减少算法的总体消耗。Zhang 和 P.Truong 分别在[3]和[4]中提出了一种实时在线近似算法，通过一个 split 阈值控制层次树规模的增长，每个达到的流记录至多引起一次计数器更新和新节点创建操作，达到减少内存占用并提高算法效率的目的。

3 问题提出

设$S=\{\alpha_1,\alpha_2,\alpha_3,\dots\}$为按照时间顺序达到的网络流，其中每一项$\alpha_i=(k_i,u_i)$由键$k_i$与一个整数$u_i\in I$

构成。$A[k]$是一组与k相关联的信号量，每个新的网络流(k_i,u_i)的到达将会引起相应信号量$A[k_i]$的更新

基金项目：本文获江苏省自然基金项目（BK2008288），国家科技支撑项目（2008BAH37B04）资助。

作者简介：江洁欣（1984～），硕士研究生，主要研究领域网络测量、网络行为学。程光（1973～），教授，硕士研究生导师，主要研究领域网络测量、网络行为学、网络管理与网络安全。

$$A[k_i] = A[k_i] + u_i$$

定义1（重聚合点）：一个流记录集合$S = \{\alpha_1, \alpha_2, \alpha_3, \dots\}$（重集），其总流量为$\mathrm{sum} = \sum_i u_i$。给定实数$\varphi (0 < \varphi \le 1)$，设$V_k = \sum_{(i:k_i=k)} u_i$表示流记录集合 S 中与$k$相关的总流量，重聚合点定义为集合$\{k | V_k \ge \varphi \mathrm{sum}\}$。

寻找重聚合点问题定义为在网络流中找出所有的 k 以及和他们相关联的流量。

定义2（层次域中的重聚合点）设输入的网络流$\{(k_i, u_i)\}$中的键k_i从层次域D中取值，层次域D的高度为h。对层次域D中的任意前缀p而言，设$elem(D,p)$为前缀p的所有子前缀的集合，令

$$V(D,p) = \sum_k V_{k:k \in elem(D,p)}$$

表示前缀p的所有流量。层次重聚合点可以定义为一系列前缀p的集合

$$\{p | V(D,p) \ge \varphi \mathrm{sum}\}$$

在五元组的流规范中，源宿 IP 地址可以自然地按照前缀划分形成层次域Dom_{ip}，端口和协议号则按照习惯简单地构成层次域Dom_{port}和Dom_{proto}。流量的一维聚合问题就是将所有流量分别在五元组流的五个维度上分解到相应的层次域中，然后通过层次域前缀之间的集合包含关系将流量聚合，最后找出所有重聚合点。

层次域的逻辑结构可以自然的用树来表达，层次树中的每个节点表示该维度上一个有意义的取值集合。根节点代表全集，它在该维度上聚合了全部流量；叶节点代表所有的独立取值，内部节点则是聚合的中间结果。如果两个节点之间存在集合包含关系，则其中的一个节点是另一个的祖先节点；否则它们不相连。

计算重聚合点问题可以通过在内存中维护完整的层次域（即使其中某个前缀可能并没有在实际流量中出现），然后遍历层次域从而找出所有流量超过阈值的前缀，这种方法对于取值空间相对很小的端口和协议号来说可以很简单的找出所有的重聚合点。但完整的 IP 地址空间大小有4GB，不可能在内存中维护整个层次域。只能维护采集到的单个流记录和它们形成的部分层次树，接着遍历层次树聚合中间节点的流量最后找出所有的重聚合点。

基于这个思想，C.Estan 在他的论文中提出了针对 IP 维度的一维流量聚合方法，算法分成两个阶段：首先遍历一次采集到的全部流记录，合并其中的重复 IP 同时创建层次树的所有叶节点，然后按照前缀关系建立内部节点并将它们的流量置为0，这个阶段完成之前 IP 前缀的层次树在内存中形成；接着对这棵层次树进行后序遍历，将每个内部节点的两个子节点流量累加赋值给该内部节点，在遍历进行的过程中就可以找出重聚合点。

算法定义 IP 前缀长度每层增加1bit，用二叉树来维护 IP 前缀之间的层次关系，前缀的变化范围从8bit 到32bit，加上根节点一共26层。算法的空间开销主要来自于维护层次树的节点，时间开销来自两个部分：节点的创建和计数器的更新。每层前缀长度增加1bit 的层次树策略存储了任意前缀长度的流量，内部

节点的数量较多，但获取内部节点流量时可以直接访问而不必计算。可以把二叉树推广到 k 叉树来存储层次树结构，相应地，每层前缀也由1bit 相应增加。

设每个节点的子节点数为$child$，聚合层数为$layer$，由于完整层次域的叶节点数等于 IP 空间大小，所以

$$child = 2^{(32/layer)}$$

可能的子节点数和相应的聚合层数一共有4种情况：

表格**1** 聚合层数和子节点数

layer	child
32	2
16	4
8	16
4	256

考虑一种更为直观的算法：对于给定的子节点数和聚合层数，对每个到达的流记录从根节点开始，依次更新直到对应叶节点路径上的$layer$个流量计数器，如果这条路径上某个节点不存在则动态建立。如图1中的示例，两个流202.112.25.69(100)和202.112.23.167(80)先后到达并依次在层次树中更新它们所对应路径上的计数器，着色节点代表该节点在流记录到达后的更新过程中被创建。

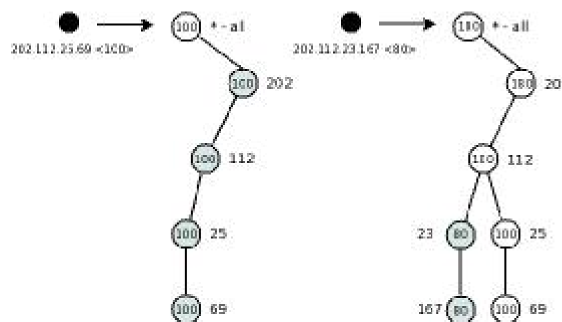

图**1** 直观算法

用华东北地区网络中心2009年7月15日9点开始采集到的5个5分钟粒度 Netflow 数据作为实验的原始数据，分别对算法的运行时间和内存占用进行实验测量，可以得到以下结论。

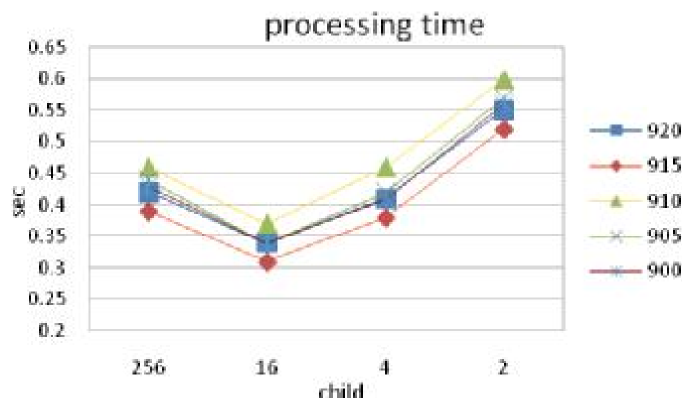

图**2** 直观算法对不同聚合层次的处理时间

算法的处理时间总体上随着聚合层数的增加呈现上升的趋势。仔细观察可以发现聚合层数从4层

（256个子节点）增加到8层（16个子节点）时，运行时间反而稍稍下降，这是因为节点动态创建所消耗的处理时间远大于更新流量计数器的开销；聚合层数从4层增加到8层时，虽然每个流记录到达需要更新的计数器数量加倍，但是聚合层数的增加引起了每个流记录更新路径上需要建立的中间节点数减少，因为其他流记录可能事先建立了这条路径上的 某些中间节点；此外在一个测量点上采集到的流量中 IP 地址往往相对集中，使得运行时间在总体上减少了。聚合层数从4增加到32的过程中，算法的运行时间并没有随着聚合层数出现指数性增长而是仅仅增加了不到50%，这也验证了节点创建操作消耗时间远大于更新计数器的消耗。通过实验数据反应出来的这两个事实发现：在基于层次树的聚合过程中，节点的创建占据了大部分的算法运行时间，流量计数器的更新操作对算法的影响很小。

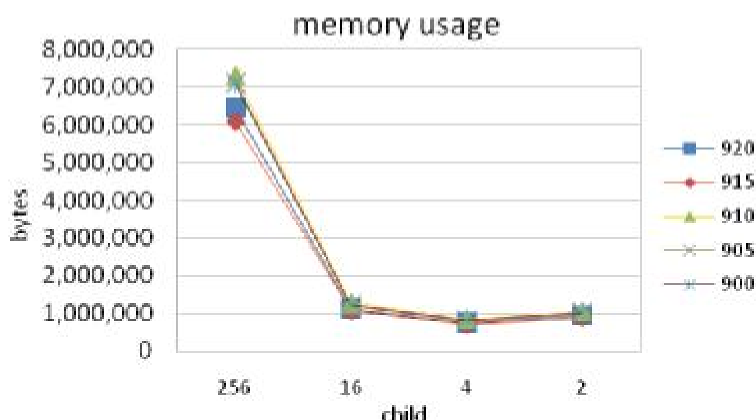

图3 直观算法内存的占用

算法的内存占用随着子节点数目总体呈现下降的趋势，虽然聚合层数减少使中间节点数量也相应下降，从而减少了层次树整体的节点数，但每个节点用于索引子节点的指针域浪费增加。对每个节点有256个子节点的情况而言，算法占用的内存中实际利用率大约只有0.8%；每个节点如果只有两个子节点，虽然中间节点数大大增加，但平均利用率超过了55%。下表是华东北地区网络中心2009年7月15日从9点到9点05分采集到的5分钟 netflow 数据，经过合并重复 IP 地址后得到31622条流记录。

表格2 算法的内存利用率

聚合层数	子节点数目	内部节点	有效指针域	利用率
4	256	27422	59043	0.008411
8	16	69732	101353	0.090842
16	4	154775	186396	0.301076
32	2	325092	356713	0.548634

C.Estan 的一维聚合算法前缀长度每层增加1bit，每个节点只有两个子节点，这可以提高内存的利用率；在聚合过程中时采用后序遍历计算中间节点的流量，而不是对每条流记录都更新它所经过路径上的所有计数器，这是基于减少更新计数器次数的考虑。访问和更新计数器的时间消耗只占算法总体运行时间的很小一部分，所以采用后续遍历期望减少计数器操作从而减少算法运行时间的策略实际带来的性能提高非常有限。其次，C.Estan 的一维聚合算法在聚合之前需要建立了所有内部节点，这带来两点问题：1.建立所有内部节点消耗了算法运行的大部分时间，虽然运用了后序遍历计算内部节点的流量减少了计数器的访问次数，算法运行的主要时间和空间开销并没有减少；2.重聚合点的数量有限，没有必要建立所有的内部节点，因为大部分的内部节点并不是重聚合点，这部分的时间和空间开销是完全没有必要的。

基于这两点考虑，本文提出一种从上到下逐层推进的一维聚合算法，算法通过避免创建不必要的内部节点，大大降低了一维聚合的总体开销。算法分成排序预处理和计算重聚合点两个部分，下面详细介绍算法。

4 算法描述

算法的层次树采用26层 IP 前缀层次树，根节点有256个子节点，分别连接到256个/8前缀；每棵/8子层次树采用每层前缀增加1bit 的二叉树结构，叶节点是/32的单个 IP，加上根节点的一层一共26层。与 C.Estan 的一维聚合算法不同的，算法从根节点出发逐层向下推进，只在层次树中维护重聚合点，算法执行完成后层次树中的所有节点即是重聚合点。

首先利用 Hash 收集流记录，目的是合并重复 IP 同时记录独立 IP 数量，这个步骤完成后根据独立 IP 数量分配内存将 Hash 表中的所有流记录转存，然后对转存后连续存放的流记录以 IP 地址作为键进行排序。排序后任意前缀的所有流记录将会连续存放，增加的排序步骤不会引起算法总运行时间的增加，这是因为无论采用自顶向下或者自底向上的的层次树建立方法，都隐含着排序：C.Estan 的一维聚合算法在建立全部叶节点后也需要对这些叶节点以 IP 地址（/32前缀）为键排序，这样才能创建中间节点建立整棵层次树；对于从上到下的建立顺序来说，当层次树建立完成之后所有叶节点已经是处于有序状态。所以，无论采用何种建立顺序，排序所有单独流记录的时间是不可避免的。通过实验也可以发现，对全部单独 IP 排序消耗的时间只占算法总时间很少一部分。

可以对排序进行一些优化，统计独立 IP 地址数量的同时可以记录它们在256个/8前缀（0.0.0.0/8～255.0.0.0/8）上的分布，在转存的过程中可以根据它们的分布，将每个独立 IP 转存到“它所在的缓存段”上：设$A_i = Count\{p|p \in prefix(i)\}$表示有多少个独立流记录（这里的流记录指合并重复 IP 流量之后的独立流记录，下同）属于 i.0.0.0/8这个前缀，$D_i = \sum_{k=0}^{i-1} A_k$表示在前缀 i.0.0.0/8之前共多少个独立的 IP，则将每个独立 IP 在D_i之后顺序插入。转存完成之后从整个流缓存上来看，所有的流记录的排列在/8前缀之间是有序的。

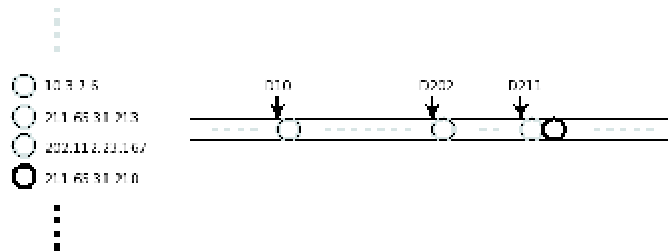

图4 分段转存独立流记录到流缓存

然后对流缓存中每个/8前缀段分段进行快速排序，前缀 i.0.0.0/8在流缓存中的所有流记录下标范围从D_i到$D_{i+1} - 1$。每个/8前缀的流记录数量为A_i，设全部独立 IP 数量为A，则$A = \sum_{i=0}^{255} A_i$。由于快速排序的时间复杂度为$O(NlogN)$，分段排序的代价为：

$$\sum_i [A_i logA_i] < \sum_i A_i * logA = AlogA$$

它小于对全部流缓存一次性排序的代价；快速排序是一种对分段有序数据敏感的排序算法，流缓存中的数据具有这种特性，这使得排序时的数据交换次数大大下降，极大提高了快速排序的性能。此外分析江苏省网边界上采集的 IP 地址/8前缀分布情况发现，只有大约不到2%的/8前缀的出现频率的总和超过50%；对于5分钟粒度的 Netflow 数据（抽样比256），平均超过70%的/8前缀在1000个以下。由于各种排序算法的效率很大程度上取决于输入数据的规模，可以针对前缀的这种分布特性根据A_i的数量采用不同的排序算法，可以进一步降低排序流记录的时间。

流记录排序完成后遍历一遍流缓存，对每段/8前缀的流记录进行处理，设$T_k^i(1 \le k \le A_i)$表示 i.0.0.0/8 前缀的第 k 个流记录的流量（如果流量不为0），将段内第 k-1个流记录的流量累加到第 k 个上。由于之前的排序已经使属于前缀的所有流记录连续有序存放，经过流量累加后，某个前缀p的流量$V(p)$就可以用$V(p) = T_e^i - T_s^i$的形式通过两次随机访问和一次减法运算得出。

$T^{202}_{99}=1000$ $T^{202}_{100}=1040$ $T^{202}_{101}=1070$ $T^{202}_{102}=1170$ $T^{202}_{103}=1225$ $T^{202}_{104}=1245$

	202.112.23.53 40	202.112.25.16 30	202.112.25.69 100	202.112.25.213 55	202.112.26.1 20

$V(202.112.25.0/24) = T^{202}_{103} - T^{202}_{100}$

图**5** 计算前缀**202.112.25.0/24**的流量

当开始计算重聚合点之前，算法一次性分配足够维护所有重聚合点的内存：层次树每层的全部节点无重叠的划分了所有流量，由于节点流量没有重叠，所以层次树中每层至多有$\lceil 1/\varphi \rceil$个重聚合点，25层（前缀从/8到/24）总共最多只可能有$25 * \lceil 1/\varphi \rceil$个重聚合点，远远小于一棵完整层次树的节点数。分配大小$[25, \lceil 1/\varphi \rceil]$的二维数组作为重聚合点缓存，另外分配长度为25的一维数组 last 记录重聚合点缓存每层上已经分配出去的节点数目。对每个重聚合点维护它的前缀、前缀长度、流缓存中对应流记录的起始结束位置、流量以及下一层的重聚合点的索引（如果存在）。

算法从根节点出发逐个检查/8前缀的流量，如果流量超过阈值则将其写入重聚合点缓存的第1层并维护该重聚合点信息，否则对该/8前缀不做处理。然后从得到的第1层所有重聚合点开始，分别找到下一层前缀长度增加1bit 后，两个子前缀在对应流缓存中的分界点并计算它们的流量将重聚合点写入重聚合点缓存；算法如此按前缀长度每次增加1bit 推进，最终计算出所有的重聚合点。由于聚合过程自顶向下推进，如果一个 IP 前缀的流量不超过阈值，那么以它为根的所有子前缀都不会再被计算，重聚合点只是层次树所有内部节点的很少一部分，所以算法的运行时间和内存占用都大大降低了。

```
1   layer = 1;
2   for i = 0 to 255
3       if(volume(D_i, D_{i+1} − 1) ≥ φsum)
4           add_hv(HH[layer, last[layer]++]);
5       endif
6   endfor
7   for layer = 1 to 25
8       child = layer + 1;
9       for current = 0 to last[layer]
10          pivot = find_pivot(HH[layer, current].s, HH[layer, current].e, child);
11          if( volume(HH[layer, current].s, pivot)≥ φsum)
12              add_hv(HH[child, last[child]++]);
13          endif
14          if( volume(pivot+1, HH[layer, current].e)≥ φsum)
15              add_hv(HH[child, last[child]++]);
16          endif
17      endfor
18  endfor
```

图6中是一个算法运行的片段的例子，假设阈值为100。10.8.0.0/28是重聚合点缓存中的已经维护的一个重聚合点，算法正要开始检查它的两个子前缀是否是重聚合点。10.8.0.0/28这个前缀在流缓存中包含了8个独立的 IP，它们各自的流量如图所示。计算10.8.0.0/28的两个子节点时，以10.8.0.7作为分界点将8个 IP 分别归属到10.8.0.0/28的两个/29子前缀中，分别计算它们各自的流量，如此逐层递进下去。

图**6** 从**10.8.0.0/28**开始计算以下的重聚合点

5 实验分析

本节利用 CERNET 江苏省网边界上采集到的数据对文章提出的快速一维聚合算法和 Estan 的一维聚合算法进行运行时间和内存占用的比较，从两方面的对比说明快速一维聚合算法通过避免建立非重聚合点带来的时间和空间效率上的提高。

实验样本是自2009年7月15日8点开始采集直到9点结束的以5分钟为测量间隔 NetflowV5格式的流量数据，总计一个小时。算法处理时间给出了一个数量上的相对关系，快速一维聚合算法的运行时间包括流记录的排序和计算重聚合点，其中绝大部分时间消耗在流记录排序上，重聚合点计算本身可以非常快的完成。

图**7** 算法运行时间对比

在内存占用方面快速一维聚合算法由于用于维护重聚合点的缓存大小是阈值百分比的函数，所以对给定的阈值是固定的；而用于维护流缓存的空间等于 Estan 自底向上聚合算法的所有叶节点，内部节点中大量非重聚合点的空间被节省了下来。

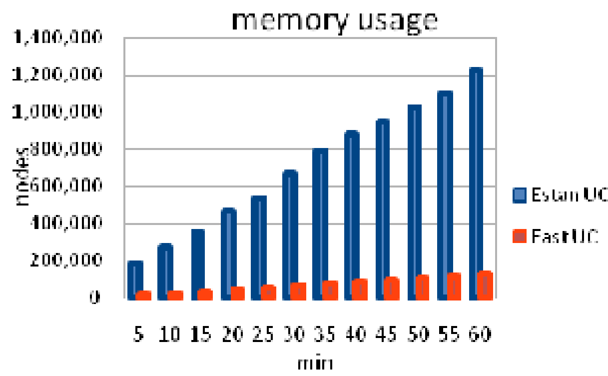

图8 算法内存占用对比

6 结论

IP 维度上的聚合是流量的一维聚合问题中最复杂的情况，IP 聚合消耗了一维聚合中最多的时间，占用了最多的资源。通过直观算法的实验发现维护层次树消耗最多计算时间的是中间节点的建立，而大量的中间节点里只有非常小一部分是的重聚合点。算法使用一次排序，通过自顶向下逐层递进的方法，完全避免了建立大量不是重聚合点的中间节点的开销，改进了一维聚合的执行效率。

参 考 文 献

[1] Cristian Estan, S. Savage, and G. Varghese. Automatically Inferring Patterns of Resource Comsumption in Network Traffic. in Proceedings of ACM SIGCOMM, 20003.

[2] Jisheng Wang, David J. Miller, and George Kesidis. Efficient Mining of the Multidimensional Traffic Cluster Hierarchy for Digesting , Visualization, and Anomaly Identification

[3] Yin Zhang, Sumeet Singh, Subhabrata Sen, Nick Duffield, and Carsten Lund. Online Indentification of Hierarchical Heavy Hitters: Algorithms, Evaluation, and Applications.

[4] Patrick Truong, Fabrice Guillemin. Dynamic Binary Tree for Hierarchical Clustering of IP Traffic.

[5] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Diamond in the rough: finding hierarchical heavy hitters in mult-dimensional data. in Proceedings of ACM SIGMOD, 2004.

A fast and memory-wise unidimensional network traffic aggregation method

JIANG Jiexin[1,2], CHENG Guang[1,2]

(1.School of Computer Science and Engineering, Southeast University, Nanjing 210096; 2.Jiangsu Province Key Laboratory of Computer Network Technology, Nanjing 210096)

Abstract Multidimensionality is the shaping characteristic that defines network traffic, which makes aggregation indispensable in order to learn the dominant patterns in traffic mix. Unidimensional aggregation is the simplest approach and it forms the foundation of multidimensional aggregation. By analyzing the major time and space cost in computing high volume clusters under unidimensional situation, we propose a fast and memory-wise hierarchical algorithm. By proceeding in a top-down fashion, our algorithm avoids the creation of clusters whose traffic volume does not exceed pre-specified fraction.

Keywords Network measurement, Traffic aggregation, Flow-mining