doi:10.11959/j.issn.1000-436x.2015268

长持续时间数据流的并行检测算法

周爱平[1,2,3]，程光[1,3]，郭晓军[1,3]，朱琛刚[1,3]

(1. 东南大学 计算机科学与工程学院，江苏 南京 210096；2. 泰州学院 计算机科学与技术学院，江苏 泰州 225300；
3. 东南大学 计算机网络和信息集成教育部重点实验室，江苏 南京 210096)

摘 要：针对现有长持续时间数据流检测算法的实时性差、检测精度与估计精度低的问题，提出长持续时间数据流的并行检测算法。基于共享数据结构的长持续时间数据流的并行检测算法中不同线程访问共享数据结构，线程之间的同步开销过大。在此基础上，基于独立数据结构的长持续时间数据流的并行检测算法中不同线程具有本地数据结构，线程之间不需要同步，产生较少的开销。理论分析与实验结果表明，基于独立数据结构的长持续时间数据流的并行检测算法具有良好的时间效率、较高的检测精度和流持续时间估计精度。

关键词：流量测量；流持续时间；数据流；并行算法

中图分类号：TP393 **文献标识码**：A

Parallel detection algorithm on long duration data streaming

ZHOU Ai-ping[1,2,3], CHENG Guang[1,3], GUO Xiao-jun[1,3], ZHU Chen-gang[1,3]

(1. School of Computer Science and Engineering, Southeast University, Nanjing 210096, China;
2. School of Computer Science and Technology, Taizhou University, Taizhou 225300, China;
3. Key Laboratory of Computer Network and Information Integration Ministry of Education, Southeast University, Nanjing 210096, China)

Abstract: Parallel data streaming algorithm was proposed according to the weak real-time performance, low detection precision and estimation accuracy for detection of long duration flow. The different threads access the shared data structure in the parallel algorithm of long duration flow detection based on shared data structure, but it generates excessive synchronous overhead. On the basis of the analytical result on the parallel algorithm of long duration flow detection with shared data structure, the different threads own local data structure in the parallel algorithm of long duration flow detection based on independent data structure, where it doesn't need synchronization and generates minor overhead. Theoretical analysis and experimental results show that the parallel algorithm of long duration flow detection based on independent data structure has good time efficiency, high detection precision and estimation accuracy of long duration flow.

Key words: traffic measurement; flow duration; data streaming; parallel algorithm

1 引言

近年来，流量测量与监控研究侧重于测量流量大小和流数。测量流量大小变化有助于诊断网络故障和检测流量异常。虽然有些网络安全事件产生大量的流数，但是可能产生较少的流量，例如端口扫描。电信运营商特别关注大流(heavy hitters)、超点(super-spreaders)检测问题。大流与产生大量流量的主机相关，应用于网络计费、异常检测及流量工程等领域[1~3]；而超点与产生大量流数的主机相关，应用于网络安全、网络管理等领域[4~6]。然而，对流持续时间[7~9]的研究相对较少。

随着互联网的快速发展和新应用不断应运而生，越来越多的应用产生长持续时间流。长持续时

收稿日期：2015-01-28；**修回日期**：2015-05-16

基金项目：国家高技术研究发展计划(“863”计划)基金资助项目(2015AA015603)；江苏省未来网络创新研究院未来网络前瞻性研究基金资助项目(BY2013095-5-03)；江苏省“六大人才高峰”高层次人才基金资助项目(2011-DZ024)

Foundation Items: The National High Technology Research and Development Program of China (863 Program) (2015AA015603); Jiangsu Future Networks Innovation Institute: Prospective Research Project on Future Networks (BY2013095-5-03); Six Talent Peaks of High Level Talents Project of Jiangsu Province (2011-DZ024)

间流广泛存在于许多互联网应用中，如网络视频、社交网络等。网络视频在服务提供商与用户之间产生大量的生命周期较长的流，且伴随大量流量；然而，社交网络上好友之间的聊天能够产生持续时间较长的流，而伴随少量流量。因此，有效利用流持续时间信息能够帮助服务提供商检测网络问题与改善用户体验。例如，通过监控网络视频应用中流持续时间估计用户的服务质量。如果大部分流是短生命周期的，那么用户在观看视频过程中存在较大的时延。服务提供商通过检测网络问题对网络链路或服务器中存在的故障做出及时的诊断和排除。另外，流持续时间的相关信息已经应用于网络流量分类[10]和异常检测[11]。僵尸网络中僵尸主机之间、僵尸主机与命令控制服务器之间维护持续时间较长的链接，为了能够接收到僵尸控制者的命令和更新信息。因此，通过长持续时间流检测有助于检测僵尸网络流量，有效阻止僵尸网络攻击。

虽然学者们在长持续时间流检测方面已经开展了一些研究，但是有限的系统计算速度与存储空间使现有长持续时间流检测算法缺乏可扩展性，主要体现在两方面：海量的网络数据流需要使用大容量存储器存储流状态；网络数据流的高速率需要使用高速存储器维护流状态。网络带宽的不断增加和互联网的普及将带来海量网络流量测量与监控难题。传统的处理器架构难以满足当前的应用需求，多核技术已成为当前处理器体系架构发展的必然趋势，基于多核的并行设计也是未来信息处理技术的趋势。多个知名厂商已经推出一系列多核处理器，如在通用多核处理器上，IBM 的 Power X Cell 8i 处理器、Sun 的 Rock 处理器、Intel 的 i7 处理器、AMD 的 Shanghai 处理器和我国的龙芯-3 处理器，它们有 4~9 个核。此外，在专用多核处理器上，Cisco 的数据分组处理器、NVIDIA 的 GTX200 线程处理器、Intel 的 Larrabee 图像处理器，它们有几十个至上百个核[12]。因此，在多核处理器计算平台上，并行技术是提高长持续时间流检测算法性能的一种有效途径。

目前，流持续时间已经成为流量测量与监控一个新的流量特征，因此，需要不同于大流、超点检测的数据流方法解决高速网络数据流持续时间检测问题。本文研究高速网络环境下在有限的存储空间内实时准确地检测长持续时间流问题。现有的长持续时间流检测方法产生误报和漏报问题，也就是将短持续时间流错误检测为长持续时间流和漏检一些长持续时间流，很难满足高速网络环境下长持续时间流检测的应用需求。

本文从共享数据结构和独立数据结构角度提出了 2 种长持续时间流的并行检测方法。基于共享数据结构的长持续时间流的并行检测方法中所有线程共享数据结构(Cuckoo 散列表)，虽然通过读写锁能够减少一部分同步开销，但是线程之间仍然存在较大的同步开销，使此方法不能有效地解决并行长持续时间流检测问题。为了克服基于共享数据结构的长持续时间流的并行检测方法的不足，基于独立数据结构的长持续时间流的并行检测方法中所有线程具有独立的本地数据结构，执行相同的检测方法，线程之间不需要同步。从复杂性角度分析基于独立数据结构的长持续时间流的并行检测方法性能，分析表明此方法具有较低的时间和空间复杂度。通过真实的网络流量对基于独立数据结构的长持续时间流的并行检测方法性能进行评价，实验结果表明，该方法在时间效率上具有较好的运行时间和加速比，并与 Chen 方法[13]、Lee[14]方法进行对比，在检测精度上不产生漏报和存在较低的误报，在流持续时间估计上具有较低的平均相对误差。因此，该方法满足高速网络长持续时间流检测的应用需求。

2　相关研究

由于高速网络环境下海量的网络流量数据和有限系统资源之间的矛盾存在，准确实时地测量和监控网络流量是一个极大的挑战。高速网络环境下准确实时地检测长持续时间流是流量测量与监控的难点，已经引起相关研究人员的关注。

目前，流持续时间研究已经取得了一些理论成果，主要体现在 2 个方面：每流持续时间检测和长持续时间流检测。Whitehead 等[15]提出了每流持续时间跟踪算法。其主要思想是利用一组 bloom filter 跟踪所有流持续时间，每个 bloom filter 表示持续时间在一定范围的流，老化过程允许 bloom filter 跟踪更长持续时间流。其难点在于跟踪所有流持续时间。Giroire 等[11]利用流持续时间检测僵尸网络，跟踪滑动窗口内每流的状态。因此，所需的内存空间是滑动窗口内流数的多倍。理论上，在足够的系统资源下每流持续时间检测算法能够获得较高的检测精度。实际上，由于系统资源的限制，每流持续时间检测算法需要折中检测精度和内存空间，以牺

牲检测精度降低占用内存空间或以占用更多内存空间提高检测精度，又无法做到实时处理。

Chen 等[14]提出了长持续时间流跟踪的数据流方法。其主要的思想是通过流的第一个分组的到达时间与最后一个分组的到达时间的间隔大于某个阈值来识别长持续时间流。然而，研究发现，攻击者在一些时间间隔内不发送分组因而能够容易逃避检测。为了检测逃避的长持续时间流，Shin 等[16]提出了基于流频数的数据流方法。其主要思想是利用一个虚拟向量估计流频数，如果流频数估计值大于$(1-\varepsilon)\theta$，其中，ε 是一个漏报因子，且 $0\leqslant\varepsilon<1$，θ 是一个给定的阈值，那么该流被检测为长持续时间流。该检测方法能够平衡误报与漏报。

Lee 等[14]提出了长持续时间流检测的数据流方法，其主要思想是利用一个计数 bloom filter 和一个 bloom filter 维护流状态，在测量时间周期开始时，初始化 bloom filter，当分组到达时，更新分组所属流对应 bloom filter 中所有位，如果该流对应计数 bloom filter 与 bloom filter 中相同位置值之和不小于一个阈值，那么该流被检测为一个长持续时间流，在测量时间周期结束时，如果 bloom filter 中位为 1，那么将计数 bloom filter 中相同位置的值增加 1，否则，置为 0。该检测方法仅需要较少的时间和内存开销，使它在网络设备和服务器中提供长持续时间流检测功能成为可能。同时，与 Chen 等[13]提出的方法相比，该方法不产生漏报，产生少量的误报。

随着骨干链路的速率提高和网络应用多样化，海量并发的活跃流存在，为了满足高速网络环境下实时准确地检测长持续时间流的应用需求，并行技术必然成为提高长持续时间流检测方法性能的一种有效途径。目前，并行技术在长持续时间流检测方面还没有开展相关研究，而其在数据挖掘[17~21]、流量监控[22~25]等领域已经取得了一些研究成果。例如，针对批处理聚类算法和流聚类算法的局限性，Hadian 等[17]提出一种并行 *k*-means 聚类算法，其主要思想是：首先将数据集划分成数据块，然后利用线程对不同的数据块并行地进行聚类，最后对每个数据块的聚类结果进行聚合。该算法充分利用多核处理器的硬件能力，通过并行处理多个数据块获得良好的性能。Das 等[18,19]提出了基于协作的加锁方法，通过一个并行流概要数据结构并行处理频繁项挖掘问题，然而，该方法的不足在于缺乏可扩展性。针对基于多核系统的网络监控应用的共同缺陷，Fusco 等[22]设计和实现多核感知流量捕获内核模块，使网络监控应用具有可扩展性。为了解决高速网络环境中大流识别问题，Zhang 等[23]利用多核处理器的并行性设计细粒度锁和精度合成两种并行算法。针对基于软件的网络设备与高端专用网络设备之间的性能差距，Buh 等[24]在多核处理器硬件平台上实现流量负载均衡。

数据流方法广泛应用于网络流量测量与监控，如大流检测[26]、超点检测[27]、流长分布估计[28,29]等。高速网络环境下网络数据流通常是高速率的，由于存在计算和内存资源的限制，只能对网络数据流处理一遍，提取网络数据流的概要信息。由于数据流方法具有一遍处理，需要较低的存储开销，满足高速网络环境的应用需求。长持续时间流可能包含少量的流量，之前提出的数据流方法(如 heavy hitter 和 super-spreaders)可能无法直接应用于长持续时间流检测。

因此，在多核处理器的并行设计与数据流技术的基础上，本文提出了 2 种长持续时间流的并行检测方法，即基于共享数据结构的长持续时间流的并行检测方法和基于独立数据结构的长持续时间流的并行检测方法。虽然前者占用较小的内存空间，但是其产生过大的同步开销，不能有效地应用于高速网络长持续时间流检测问题。后者具有独立的本地数据结构，通过相同的检测算法独立检测长持续时间流。理论分析表明该方法具有较低的时间和空间复杂度；实验评价表明该方法具有良好的时间效率和较高的检测精度和估计精度。

3 问题定义和数据结构

3.1 问题定义

假设网络数据流 $S = p_1,\cdots,p_t,\cdots,p_n$ 被均匀划分为 R 个子流 $S_1, S_2,\cdots,S_R$，子流 S_l $(1 \leqslant l \leqslant R)$为由二元组$(f_{\mathrm{id}}, t)$构成的时间序列，$f_{\mathrm{id}}$ 是流标识，t 是分组的到达时间。因此，网络数据流 S 是由子流 S_1, $S_2,\cdots,S_R$ 依据分组的到达时间 t 重新排列后得到的时间序列。

定义 1 网络数据流(network data stream)。在一定的时间区间内顺序到达的分组序列，则称为网络数据流，记为 $S = p_1,\cdots,p_i,\cdots,p_n$。其中，$p_i = (f_{\mathrm{id}}, t)$ 表示第 i 个分组信息，f_{id} 表示流标识，由源/目的 IP 地址、源/目的端口及协议五元组构成，t 表示分组的到达时间。

定义 2　活跃流(active flow)。在一定的时间区间内属于流的第一个分组到达后每个超时间隔内至少有一个其分组到达，则称为活跃流。

定义 3　流持续时间(flow duration)。在一定的时间区间内满足超时的流第一个分组到达时间与其最后一个分组到达时间之间的差值，则称为流持续时间，记为 f_d。如图 1 所示，流的第一个分组在第一个 ΔT 时间内到达，终止于第 7 个 ΔT 时间内，流的持续时间为 $6\Delta T$。

图 1　流持续时间的示例

对并行长持续时间流检测问题可以进行如下定义。设长时间持续流的准确率为$\phi\in(0, 1)$，误差率为 $\varepsilon\in(0, 1)$ ($\varepsilon<<\phi$)，当用户发送长持续时间流查询请求时，如果查询结果满足以下 3 个条件：

1) 输出持续时间 $f_d> \phi D$ 的所有流，D 表示测量时间区间内所有流的持续时间总和；

2) 不输出持续时间 $f_d < (\phi-\varepsilon)D$ 的任何流；

3) 流持续时间的估计值与真实值之间的误差不大于 εD。

则称 f_d 为长持续时间流，记为 LDF(long duration flow)。

依据 LDF 的定义，在 LDF 检测中参数ϕ、ε 的取值大小是关键的。当参数 ϕ 取值较小时，可能有更多的流被识别为 LDF，从而需要较大的内存空间检测 LDF；当参数 ϕ 取值较大时，可能有更多 LDF 漏掉。另外，参数 ε 取值大小也会影响 LDF 检测精度。因此，根据实际的网络应用需求，设置参数 ϕ、ε 的大小。

为了能够满足高速网络流持续时间检测的应用需求，充分利用多核计算平台的并行性，设计并行的流持续时间检测算法成为一种必然选择。

3.2　数据结构

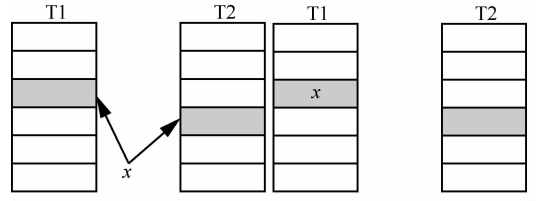
Cuckoo 散列表[30]是 LDF 检测方法的基本数据结构。Cuckoo 散列是一种散列冲突解决方法，其目的是即使使用简易的散列函数能够实现流的均匀分布，同时保证 $O(1)$的流查询时间。其基本思想是使用 2 个散列函数来处理冲突，从而每个流对应 2 个位置。Cuckoo 散列表中每个数据项是一个四元组的元素，记为(f_{id}, c_f, e_f, t_f)，其中，f_{id} 表示流标识，c_f 表示流持续时间，e_f 表示误差，t_f 表示时间戳。Cuckoo 散列算法如图 2 所示，具体步骤为：

1) 利用散列函数 func1、func2 计算流 x 在散列表 T1、T2 中的位置；

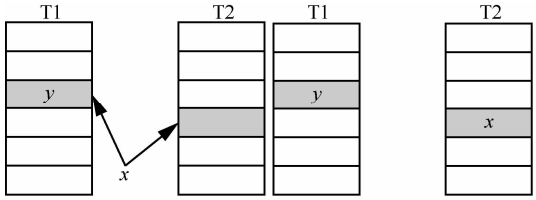
2) 如果流 x 对应散列表的 2 个位置均为空，那么将流 x 插入其中任意一个位置，如图 2(a)所示；

3) 如果流 x 对应散列表的 2 个位置中其一为空，那么将流 x 插入为空的位置，如图 2(b)所示；

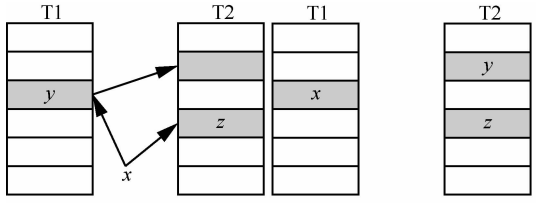
4) 如果流 x 对应散列表的 2 个位置均为非空，那么任意选择一个位置，将该位置的流 y 踢出，然后插入流 x；如果流 x 插入散列表 T1，那么利用散列函数 func2 计算流 y 在散列表 T2 中的位置，如果散列表 T2 中的位置为空，那么将流 y 插入散列表 T2，否则将该位置的流踢出，然后插入流 y，被踢出的流需要重新插入，直到没有其他流被踢出为止，如图 2(c)所示。



(a) 2 个位置为空



(b) 其中一个位置为空



(c) 2 个位置为非空

图 2　Cuckoo 散列算法

当 Cuckoo 散列表满时，需要建立一个以流持续时间为关键字的最小堆，使每次从无序的流记录中选择最小持续时间流的效率高、复杂度低。最小堆是一棵完全二叉树，其任何一个父节点的流持续时间不大于其左右孩子节点的流持续时间。设流记录序列为$\{A[i]|\ 0 \leqslant i \leqslant n-1\}$，最小堆中节点之间的关系表示为

$$A[i]f_{\mathrm{d}} \leqslant A[2i+1]f_{\mathrm{d}} \,\&\&\, A[i]f_{\mathrm{d}} \leqslant A[2i+2]f_{\mathrm{d}} \quad (1)$$

由此可知，最小堆的堆顶元素是流持续时间最小的流记录。

最小堆排序的基本思想如下。

1) 将初始流记录序列$\{A[i]\mid 0 \leqslant i \leqslant n-1\}$建成一个最小堆。

2) 将流持续时间最小的流记录$A[0]$和最后一个流记录$A[n-1]$交换，从而得到新的流记录序列$\{A[i]\mid 0 \leqslant i \leqslant n-2\}$和有序的流记录序列$\{A[n-1]\}$，且满足$A[i]f_{\mathrm{d}} \geqslant A[n-1]f_{\mathrm{d}}\ (0 \leqslant i \leqslant n-2)$。

3) 因为新的堆顶流记录$A[0]$可能不满足最小堆关系，所以需要将新的流记录序列$\{A[i]\mid 0 \leqslant i \leqslant n-2\}$重新调整为最小堆。

4) 将流持续时间最小的流记录$A[0]$和最后一个流记录$A[n-2]$交换，由此得到新的流记录序列$\{A[i]\mid 0 \leqslant i \leqslant n-3\}$和有序的流记录序列$\{A[i]\mid n-2 \leqslant i \leqslant n-1\}$，且满足$A[i].f_{\mathrm{d}} \geqslant A[n-2].f_{\mathrm{d}}(0 \leqslant i \leqslant n-3)$。

5) 因为新的堆顶流记录$A[0]$同样可能不满足最小堆关系，所以仍需要将$\{A[i]\mid 0 \leqslant i \leqslant n-3\}$重新调整为最小堆，直到此流记录序列中只有一个元素为止，最小堆排序过程完成。

4 基于共享数据结构的 LDF 并行检测方法

本节描述基于共享数据结构的 LDF 并行检测方法，其框架如图 3 所示。该方法中所有线程共享数据结构(Cuckoo 散列表)。基于共享数据结构的 LDF 并行检测方法中，每个子流被分配到不同的线程，当多个线程需要同时访问共享数据结构时，需要对不同的表项进行加锁实现同步，为了减少通过加锁产生的同步开销，此外，散列表的访问过程中读操作远比写操作频繁，因此，通过引入读写锁实现线程之间的同步。

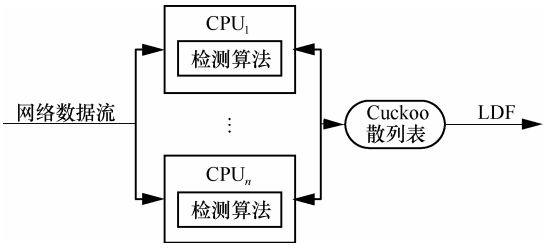


图 3 共享数据结构的框架

当线程读取 Cuckoo 散列表中对应表项时，需要以读模式对它加锁，所有试图以读模式对它进行加锁的线程均可以访问该表项，然而，如果线程以写模式对此表项进行加锁，那么读写锁通常会阻塞后续的读模式锁请求，因此，可以避免读模式锁长期占用，而等待的写模式锁请求长期阻塞。

基于共享数据结构的 LDF 并行检测方法中，所有线程维护和更新共享数据结构(Cuckoo 散列表)中每个流记录，当所有线程执行完成后，进行 LDF 检测，测量时间周期结束后，输出所有 LDF。该方法的具体流程如下。

1) 初始化 Cuckoo 散列表、测量时间区间的开始时间t_{s}、流持续时间c'、时间戳t'及流超时记录标记*flag*。

2) 当分组到达时，首先需要将其时间戳转换为分组到达的时间区间，转换表达式为

$$t = \left\lceil \frac{T}{\Delta T} \right\rceil \quad (2)$$

其中，T表示分组的时间戳，ΔT表示超时。

3) 当每个测量时间区间开始时，利用流超时标记判断超时流记录是否被删除，如果没有删除超时流记录，那么扫描整个 Cuckoo 散列表并删除超时流记录，然后，更新流超时记录标记 *flag*。另外，如果不属于 Cuckoo 散列表的最大持续时间流发生超时，那么将该流记录的持续时间c'置为 0，同时将测量时间区间的开始时间t_{s}置为下一个测量时间区间的开始时间。

4) 如果到达分组的所属流存在于 Cuckoo 散列表中，分 2 种情况讨论。当该流记录的时间戳与分组到达的时间区间相等，表明在当前的时间区间内已经更新了该流记录的时间戳，从而跳过到达分组；当该流记录的时间戳与分组到达的时间区间不相等，表明在前一个时间区间内已经更新了该流记录的时间戳，而在当前的时间区间内还没有更新该流记录的时间戳。因此，需要更新该流记录的持续时间c_{f}与时间戳t_{f}，更新表达式为

$$c_{\mathrm{f}} = c_{\mathrm{f}} + 1 \quad (3)$$

$$t_{\mathrm{f}} = t \quad (4)$$

5) 如果到达分组的所属流不存在于 Cuckoo 散列表中，且不属于 Cuckoo 散列表中最大持续时间流的持续时间为 0 或最大持续时间流的时间戳属于前一个时间区间，那么将该流插入到 Cuckoo 散列表中，同时对该流的持续时间c_{f}、误差e_{f}及时间戳t_{f}进行相应的更新，更新表达式为

$$c_{\mathrm{f}} = c' + 1 \tag{5}$$

$$e_{\mathrm{f}} = c' \tag{6}$$

$$t_{\mathrm{f}} = t \tag{7}$$

6) 当 Cuckoo 散列表已满时，通过最小堆排序获得最小持续时间流，查找 Cuckoo 散列表中最小持续时间流，然后删除 Cuckoo 散列表中该流记录，同时更新不属于 Cuckoo 散列表且具有最小持续时间流的持续时间 c' 与时间戳 t'，更新表达式为

$$c' = c_{\mathrm{f}_{\min}} \tag{8}$$

$$t' = t \tag{9}$$

7) 如果到达分组所属的流不存在于 Cuckoo 散列表中，且该流的持续时间 c_{f} 与误差 e_{f} 的差值大于阈值 d，那么将该流检测为 LDF。

在运行时间和加速比方面，通过实际的网络流量数据对基于共享数据结构的 LDF 并行检测方法进行评价。

实验在一台服务器上进行。服务器的配置信息为：一个英特尔至强 4 核处理器(E5-2403 1.80 GHz)、8 GB 内存及 CentOS 6.5 操作系统。CERNET 数据集[31]采集于中国教育和科研计算网的边界路由器，仅包含匿名的分组头部信息，持续时间为 30 min，用 Trace1 表示 CERNET 数据集。CAIDA 数据集[32]采集于 OC 48 骨干链路，仅包含匿名的分组头部信息，持续时间为 30 min，用 Trace2 表示 CAIDA 数据集。表 1 显示了网络流量的部分统计信息，包括网络流量的总字节数、分组数及流数。通过 CERNET 数据集计算流持续时间的累积概率分布(CDF)，如图 4 所示。79.9%流的持续时间小于 2 s，19.8%流的持续时间不小于 15 min，0.3%流的持续时间大于 15 min[7]。

表 1　　数据集的统计信息

数据集	字节数	分组数	流数	持续时间/min
Trace1	1 958 084 711	29 413 424	2 852 574	30
Trace2	6 394 510 148	135 923 623	9 707 390	30

图 5 显示了基于共享数据结构的 LDF 并行检测方法在 CERNET 数据集和 CAIDA 数据集上的运行时间。其中，横轴表示检测方法中线程数量，纵轴表示检测方法的运行时间。从图 5 可知，随着线程数量的增加，该方法的运行时间增加，然后渐渐地减少。图 6 显示了基于共享数据结构的 LDF 并行检测方法在 CERNET 数据集和 CAIDA 数据集上的加速比。横轴表示检测方法中线程数量，纵轴表示单线程的 LDF 检测方法与该方法的运行时间的比值。从图 6 可知，随着线程数量的增加，该方法的加速比先减小，然后渐渐地增加。

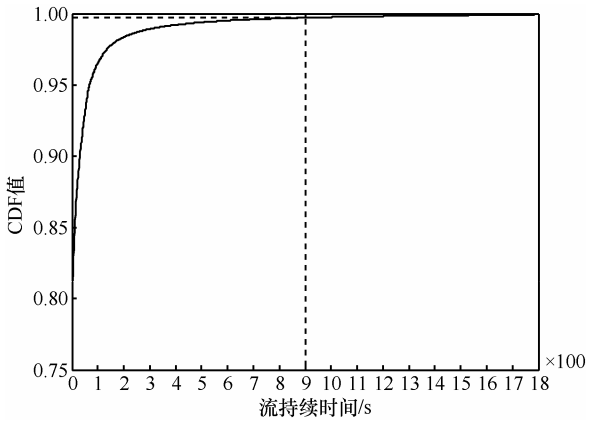


图 4　流持续时间的累积概率分布

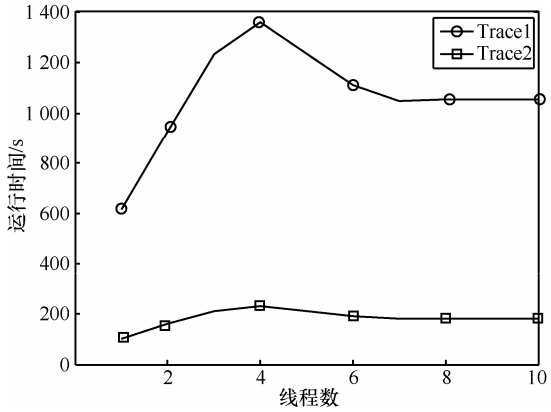


图 5　LDF 检测的运行时间

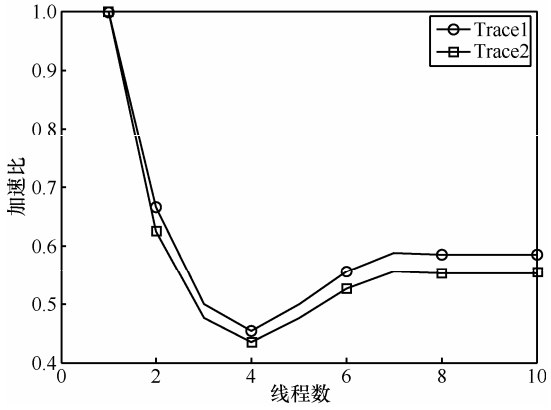


图 6　LDF 检测的加速比

从上述实验结果分析可知，基于共享数据结构的 LDF 并行检测方法中线程数量的增加反而使得该方法的性能下降。可能原因是线程在处理每个子

流时需要同时获得多把锁，导致多个线程访问共享资源时需要等待，同步开销过大。同时，还可以发现，当线程的数量少于 4 时，该方法的性能会随着线程的增加而下降；而当线程的数量多于 4 时，算法的性能慢慢增加，然后趋于稳定。造成这样现象的主要原因是服务器共有 4 个核，4 个线程分别在每个核上独立地运行是最理想的。所以，当线程的数量少于 4 时，线程应该能够实现真正的并行运行，但线程之间的同步使得开销过大。

综上所述，基于共享数据结构的 LDF 并行检测方法仅需要访问共享数据结构，从而可以显著地减少多个数据结构的合并开销，同时也节省了内存空间。然而，当分组到达时，每个线程访问共享数据结构，为了实现多个线程之间的同步，引入读写锁对共享数据结构的表项进行加锁。虽然读写锁能一定程度上提高该方法的并发性，但是多核处理器系统允许多个读者同时访问共享资源，写者是排他性的，一个读写锁同时只能有一个写者或多个读者。基于共享数据结构的 LDF 并行检测方法中，多个线程竞争和等待共享数据结构(Cuckoo 散列表)，产生多大的同步开销。因此，基于共享数据结构的 LDF 并行检测方法无法满足高速网络长持续时间流检测的应用需求。

5 基于独立数据结构的 LDF 并行检测方法

本节从方法描述、方法流程、性能分析及实验评价方面阐述基于独立数据结构的 LDF 并行检测方法。该方法中所有线程具有相同的本地数据结构(Cuckoo 散列表)，执行相同的检测算法。

5.1 方法描述

为了能够适应流持续时间检测的应用需求，将属于相同流的分组划分到相同的子流，由于流长分布具有重尾特性，如果按照流数划分子流，可能造成有些线程的负载过重，为了实现不同线程之间的负载均衡，因此，假设每个线程处理大致相同数量的分组。由于基于共享数据结构的 LDF 并行检测方法中不同线程之间需要同步，导致同步开销过大，无法满足高速网络长持续时间流检测的应用需求。为了克服基于共享数据结构的 LDF 并行检测方法的不足，本文提出了基于独立数据结构的 LDF 并行检测方法。该方法中不同线程之间不需要同步，每个线程具有独立的本地数据结构，执行相同的检测算法，处理一个子流，维护一个本地数据结构(Cuckoo 散列表)，独立进行 LDF 检测。

基于独立数据结构的 LDF 并行检测方法的框架如图 7 所示。与基于共享数据结构的 LDF 并行检测方法相比，基于独立数据结构的 LDF 并行检测方法的不同之处在于每个线程只需更新本地数据结构，独立检测 LDF，不产生同步开销和汇聚开销。当所有线程执行完成后，所有线程的检测结果的并集将为基于独立数据结构的 LDF 并行检测方法的最终检测结果。因此，该方法能够满足高速网络长持续时间流检测的应用需求。

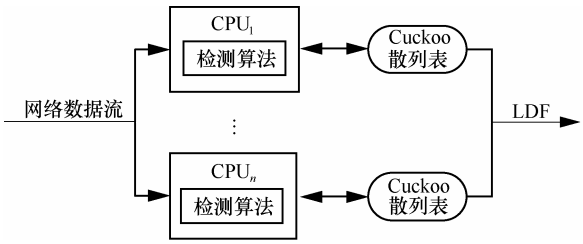


图 7 独立数据结构的框架

5.2 方法流程

基于独立数据结构的 LDF 并行检测方法的基本过程：当网络数据流到达时，首先计算分组到达的时间区间，如果分组到达的时间戳与测量时间周期开始时间属于不相等的时间区间，那么移走超时流；当分组所属的流存在于 Cuckoo 散列表中时，如果该流记录的时间戳属于当前分组到达的时间区间的前一个时间区间，那么更新该流记录的持续时间和时间戳；当分组所属的流不存在于 Cuckoo 散列表中时，如果不属于 Cuckoo 散列表的流记录的最大流持续时间等于零或相应的时间戳属于当前分组到达的时间区间的前一个时间区间，那么将该流插入到 Cuckoo 散列表；当 Cuckoo 散列表已满时，创建最小堆，使用最小堆删除 Cuckoo 散列表中最小持续时间的流记录；当分组所属的流存在于 Cuckoo 散列表中且流记录的持续时间与误差的差值大于阈值时，该流被识别为 LDF；最后，将各个 LDF 子集的并集作为检测结果。

基于独立数据结构的 LDF 并行检测方法流程如下。

1) 初始化 Cuckoo 散列表、测量时间区间开始时间 t_s、流持续时间 c' 和时间戳 t'

2) 当属于流 f_{id} 的某个分组到达时，提取时间戳 T，计算其到达的时间区间 t

3) if $t \neq t_s$

4)　移走 Cuckoo 散列表中超时的流记录

5)　if $t' \neq t_s$

6)　　将 c' 置为 0

7)　将 t_s 赋值为 t

8) if 流 f_{id} 属于 Cuckoo 散列表

9)　if $t_f = t-1$

10)　　更新流持续时间 c_f 和时间戳 t_f

11) else if $c' = 0$ 或 $t' = t-1$

12)　　在 Cuckoo 散列表中创建新的流记录，并更新流持续时间 c_f、误差 e_f 和时间戳 t_f

13) if Cuckoo 散列表已满

14)　建立最小堆，删除 Cuckoo 散列表中持续时间最短的流记录，并更新流持续时间 c' 和时间戳 t'

15) if 流 f_{id} 属于 Cuckoo 散列表且流持续 c_f 与误差 e_f 的差值大于阈值

16)　　识别流 f_{id} 为长持续时间流

17) 合并各个 LDF 子集

5.3　性能分析

定理 1　假设 n 个子流，基于独立数据结构的 LDF 并行检测方法处理每个分组的平均时间复杂度为 $O\left(\frac{1}{n}\right)$。

证明　当分组到达时，计算分组到达的时间区间 t，查询 Cuckoo 散列表中 2 个位置，如果分组所属的流 f_{id} 存在于 Cuckoo 散列表中，然后将该流记录的时间戳 t_f 与当前分组到达的时间区间 t 进行比较，若 t_f 与 t 相等，说明在当前的时间区间内该流记录的时间戳 t_f 已经被更新，则跳过该分组，否则，说明该流记录的时间戳 t_f 在前一个时间区间内被更新了，而在当前的时间区间内还没有被更新，因此，需要更新该流记录的持续时间 c_f 和相应的时间戳 t_f，那么该更新过程能够在 $O(1)$时间内完成；如果分组所属的流不存在于 Cuckoo 散列表中，根据流持续时间 c' 和时间戳 t' 判断是否将该流插入到 Cuckoo 散列表中，如果不属于 Cuckoo 散列表且最小持续时间流的持续时间 c' 为 0 或相应的时间戳 t' 属于上一个时间区间，那么需要将该流插入 Cuckoo 散列表和更新其持续时间 c_f、误差 e_f 及时间戳 t_f，该过程同样能够在 $O(1)$时间内完成。假设 n 个子流，因此，基于独立数据结构的 LDF 检测的并行数据流方法处理每个分组的平均时间复杂度为 $O\left(\frac{1}{b}\right)$ 得证。

定理 2　假设 n 个子流，$m = \left\lceil \frac{1}{\varepsilon} \right\rceil$，在每个测量时间区间内，基于独立数据结构的 LDF 并行检测方法维护 Cuckoo 散列表的平均时间复杂度为 $O\left(\frac{m}{n}\right)$。

证明　当每个测量时间区间开始时，扫描整个 Cuckoo 散列表，根据分组到达的时间区间 t 和流记录的时间戳 t_f 删除超时的流，该过程在 $O\left(\frac{1}{\varepsilon}\right)$ 时间内完成。如果 Cuckoo 散列表已满，删除最小持续时间流记录，同时更新不属于 Cuckoo 散列表且最小持续时间流的持续时间 c' 和时间戳 t'，该过程也在 $O\left(\frac{1}{\varepsilon}\right)$ 时间内完成。因此，在每个时间区间内，维护 Cuckoo 散列表的时间复杂度为 $O\left(\frac{1}{\varepsilon}\right)$。假设 n 个子流，$m = \left\lceil \frac{1}{\varepsilon} \right\rceil$，所以在每个测量时间区间内，维护 Cuckoo 散列表的平均时间复杂度为 $O\left(\frac{m}{n}\right)$ 得证。

定理 3　假设 n 个子流，$m = \left\lceil \frac{1}{\varepsilon} \right\rceil$，基于独立数据结构的 LDF 并行检测方法的平均空间复杂度为 $O(nm)$。

证明　如果在每个时间区间内每个流只有一个分组出现，LDF 检测问题等价于大流识别问题。基于大流识别的分析结果，至少需要 $O\left(\frac{1}{\varepsilon}\right)$ 内存空间检测 LDF。在一般的情况下，在每个时间区间内只考虑每个流的第一个分组，通过流标识 f_{id} 和时间戳 t_f 跳过之后出现的分组。因此，基于独立数据结构的 LDF 并行检测方法的空间复杂度为 $O\left(\frac{1}{\varepsilon}\right)$。假设 n 个子流，$m = \left\lceil \frac{1}{\varepsilon} \right\rceil$，因此，基于独立数据结构的 LDF 并行检测方法的平均空间复杂度为 $O(nm)$得证。

5.4　实验评价

前面对基于独立数据结构的 LDF 并行检测方法进行了理论分析，本节利用相同的实验环境和数据集对基于独立数据结构的 LDF 并行检测方法进行评价。从运行时间、检测精度及流持续

时间估计方面评价该方法的性能，并与相关方法进行比较。令 ILDF 表示基于独立数据结构的 LDF 并行检测方法，Chen 表示 Chen 等[13]提出的 LDF 检测方法，Lee 表示 Lee 等[14]提出的 LDF 检测方法。

1) 评价标准

实验中，利用运行时间和加速比评价时间效率。加速比指同一个任务在单处理器系统和并行处理器系统中运行消耗的时间比率，用于衡量基于独立数据结构的 LDF 并行检测方法的性能。

通过误报率和漏报率评价 LDF 检测精度。误报率(false positive rate)指被错误识别的 LDF 数量与被识别的 LDF 总数的比率。漏报率(false negative rate)指没有被识别的 LDF 数量与实际的 LDF 数的比率。令实际的 LDF 集为 A，被识别的 LDF 集为 B，误报率表示为

$$false\ \ positive\ \ rate = \frac{|B - A|}{|B|} \tag{10}$$

漏报率表示为

$$false\ \ negative\ \ rate = \frac{|A - B|}{|A|} \tag{11}$$

误报率、漏报率越小表明检测精度越高。

通过相对误差评价流持续时间。相对误差(relative error)用于衡量流持续时间的测量值与真实值之间的差异程度。令流持续时间的测量值为 $\hat{F}$ 与真实值为 F，则流持续时间相对误差表示为

$$relative\ \ error = \frac{|\hat{F} - F|}{F} \tag{12}$$

相对误差越小表明测量值越精确。

2) 时间效率

图 8 显示了基于独立数据结构的 LDF 检测的并行检测方法在 CERNET、CAIDA 数据集上的运行时间。从图 8 可知，随着线程数量的增加，此方法的运行时间先减小后增加。图 9 显示了基于独立数据结构的 LDF 并行检测方法的加速比。从图 9 可知，随着线程数量的增加，此方法的加速比先增加后减小。从上述实验结果可以看出，因为服务器只有 4 个核，所以随着线程数量的增加，基于独立数据结构的 LDF 并行检测方法的性能先提高后下降。

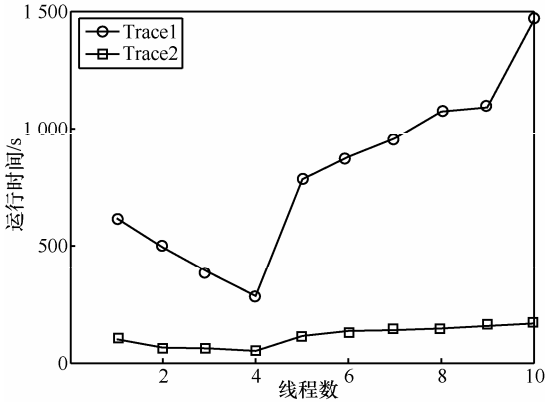


图 8 LDF 检测的运行时间

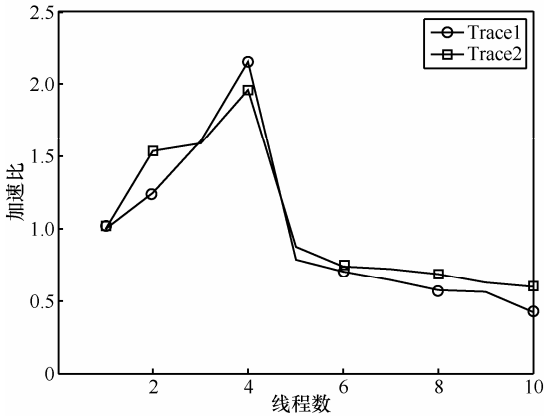


图 9 LDF 检测的加速比

3) 检测精度

图 10 显示了 3 种 LDF 检测方法的误报率。从图 10 可知，随着内存开销增加，Chen 与 Lee 方法的误报率减小，而 ILDF 方法不产生误报。图 11 显示了 3 种 LDF 检测方法的漏报率。从图 11 可知，随着内存开销增加，Chen 与 ILDF 方法的漏报率减小，而 Lee 方法不产生漏报。从上述实验结果可知，在误报率和漏报率方面，ILDF 方法的检测精度优于 Chen、Lee 这 2 种方法。

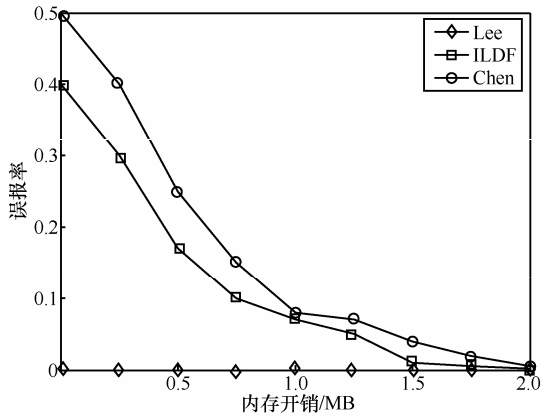


图 10 LDF 检测的误报率

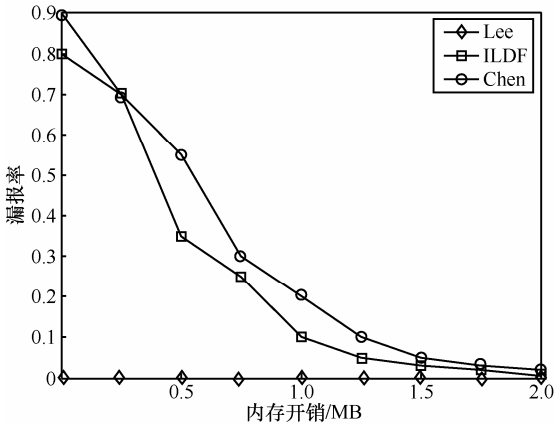


图 11　LDF 检测的漏报率

4) 流持续时间估计

Chen 方法中，流持续时间估计表示为

$$\hat{d}_f^1 = B_i(f) = \min_{1\leqslant k\leqslant K}\left(B_i\left[h_{ik}(f)\right]\right), i=1,2 \qquad (13)$$

Lee 方法中，流持续时间估计表示为

$$\hat{d}_f^2 = \min_{1\leqslant k\leqslant K}\left(C\left[H_k(f)\right]+B\left[H_k(f)\right]\right) \qquad (14)$$

ILDF 算法中，流持续时间估计表示为

$$\hat{d}_{\mathrm{f}}^3 = \begin{cases} c_{\mathrm{f}} - e_{\mathrm{f}}, \text{ 如果流}f\text{属于Cuckoo 散列表} \\ 0, \text{ 其他} \end{cases} \qquad (15)$$

流持续时间的相对误差表示为

$$relative\ \ error = \frac{\left|\hat{d}_{\mathrm{f}}^i - d_{\mathrm{f}}\right|}{d_{\mathrm{f}}},\ i=1,2,3 \qquad (16)$$

图 12 显示了流持续时间的平均相对误差。从图 12 可知，ILDF 方法中，流持续时间的平均相对误差开始相对比较稳定，由于短持续时间流不存在于 Cuckoo 散列表，流持续时间估计均设为 0。由于流持续时间在[$(\phi-\varepsilon)D$, ϕD]范围内存在漏报，在

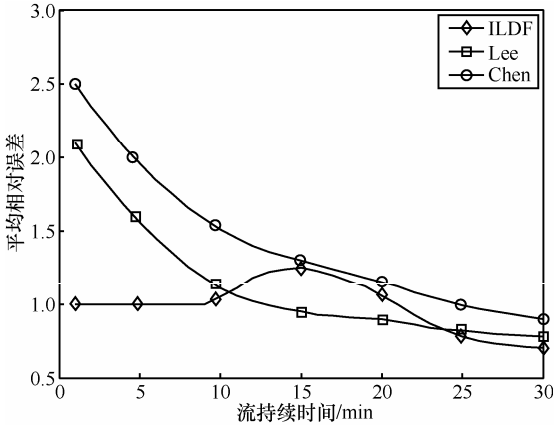


图 12　流持续时间的平均相对误差

15 min 附近流持续时间的相对误差总体上是增加的，此后，流持续时间的相对误差呈现下降趋势。总之，与 Chen、Lee 方法相比，ILDF 算法获得最优的流持续时间估计。

6　结束语

本文研究了在多核处理器硬件平台上的长持续时间流并行检测问题。从共享数据结构和独立数据结构两方面设计了长持续时间流并行检测算法。基于共享数据结构的长持续时间流检测算法存在资源共享，导致线程之间存在较大的竞争开销，并不能有效地解决长持续时间流并行检测问题；而基于独立数据结构的长持续时间流检测算法中每个线程具有独立的本地数据结构，线程之间不存在资源共享，因此，不产生竞争开销，并且没有合并开销。利用真实的网络流量进行实验，实验结果表明 2 种长持续时间流并行检测算法存在不同的优缺点，基于共享数据结构的长持续时间流检测算法具有占用内存空间小的优点，其不足之处在于执行时间长，竞争开销过大；基于独立数据结构的长持续时间流检测算法具有时间效率高和检测精度高，满足高速网络流量监测的应用需求，其不足之处在于占用稍大的内存空间。总体上，基于独立数据结构的长持续时间流并行检测算法优于基于共享数据结构的长持续时间流并行检测算法，这 2 种算法的有效结合将成为下一步研究的重点。

参考文献：

[1] 王宏, 龚正虎. Hits 和 Holds: 识别大象流的两种算法[J]. 软件学报, 2010, 21(6): 1391-1403.
WANG H, GONG Z. Hits and Holds: two algorithms for identifying the elephant flows[J]. Journal of Software, 2010, 21(6): 1391-1403.

[2] 张震, 汪斌强, 张风雨, 等. 基于 LRU-BF 策略的网络流量测量算法[J]. 通信学报, 2013, 34(1): 111-120.
ZHANG Z, WANG B Q, ZHANG F Y, *et al*. Traffic measurement algorithm based on least recent used and Bloom filter[J]. Journal on Communications, 2013, 34(1): 111-120.

[3] FREDERIC R C. Scalable identification and measurement of heavy-hitters[J]. Computer Communications, 2013, 36(8): 908-926.

[4] SHIN S, IM E, YOON M. A grand spread estimator using a graphics processing unit[J]. Journal of Parallel and Distributed Computing, 2014, 74(2): 2039-2047.

[5] LIU Y, CHEN W, GUAN Y. Identifying high-cardinality hosts from network-wide traffic measurements[A]. Proceedings of the IEEE Conference on Communications and Network Security (CNS)[C]. National Harbor, MD, USA, 2013. 287-295.

[6] CHENG G, TANG Y. Line speed accurate superspreader identification

using dynamic error compensation[J]. Computer Communications, 2013, 36(13): 1460-1470.

[7] BROWNLEE N, CLAFFY K. Understanding Internet traffic streams: dragonflies and tortoises[J]. IEEE Communications Magazine, 2002, 40(10): 110-117.

[8] LEE D J, BROWNLEE N. Passive measurement of one-way and two-way flow lifetimes[J]. ACM SIGCOMM Computer Communication Review, 2007, 37(3): 17-28.

[9] QUAN L, HEIDEMANN J. On the characteristics and reasons of long-lived internet flows[A]. Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC)[C]. New York, USA, 2010. 444-450.

[10] LI D, Hu G, Wang Y, *et al*. Network traffic classification via non-convex multi-task feature learning[J]. Neurocomputing, 2015, 152(25): 322-332.

[11] GIROIRE F, CHANDRASHEKAR J, TAFT N, *et al*. Exploiting temporal persistence to detect covert botnet channels[A]. Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID)[C]. Saint-Malo, France, 2009. 326-345.

[12] http://tech.sina.com.cn/h/2009-04-09/13372987082.shtml[EB/OL].

[13] CHEN A, JIN Y, CAO J, *et al*. Tracking long duration flows in network traffic[A]. Proceedings of the International Conference on Computer Communications (INFOCOM)[C]. San Diego, USA, 2010. 1-5.

[14] LEE S, SHIN S, YOON M. Detecting long duration flows without false negatives[J]. IEICE Transactions on Communications, 2011, E94-B(5): 1460-1462.

[15] WHITEHEAD B, LUNG C H, RABINOVITCH P. An efficient hybrid approach to per-flow state tracking for high-speed networks[J]. Computer Communications, 2013, 36(8): 927-938.

[16] SHIN S, YOON M. Virtual vectors and network traffic analysis[J]. IEEE Network, 2012, 26(1): 22-26.

[17] HADIAN A, SHAHRIVARI S. High performance parallel *k*-means clustering for disk-resident datasets on multi-core CPUs[J]. The Journal of Supercomputing, 2014, 69(2): 845-863.

[18] DAS S, ANTONY S, AGRAWAL D, *et al*. CoTS: a scalable framework for parallelizing frequency counting over data streams[A]. Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE)[C]. Shanghai, China, 2009. 1323-1326.

[19] DAS S, ANTONY S, AGRAWAL D, *et al*. Thread cooperation in multicore architectures for frequency counting over multiple data streams[J]. Proceedings of the VLDB Endowment, 2009, 2(1): 217-228.

[20] TANGWONGSAN K, TIRTHAPURA S, WU K. Parallel streaming frequency-based aggregates[A]. Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)[C]. Prague, Czech Republic, 2014. 236-245.

[21] ROY P, TEUBNER J, ALONSO G. Efficient frequent item counting in multi-core hardware[A]. Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)[C]. Beijing, China, 2012. 1451-1259.

[22] FUSCO F, LUCA D. High speed network traffic analysis with commodity multi-core systems[A]. Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC)[C]. Melbourne, Australia, 2010. 218-224.

[23] ZHANG Y, FANG B, ZHANG Y. Parallelizing weighted frequency counting in high-speed network monitoring[J]. Computer Communications, 2011, 34(4): 536-547.

[24] BUH T, TROBEC R, CIGLIC A. Adaptive network-traffic balancing on multi-core software networking devices[J]. Computer Networks, 2014, 69(20): 19-34.

[25] ZHANIKEEV M. A lockfree shared memory design for high-throughput multicore packet traffic capture[J]. International Journal of Network Management, 2014, 24(4): 304-317.

[26] ZHANG Y, FANG B, ZHANG Y. Identifying heavy hitters in high-speed network monitoring[J]. Science China: Information Sciences, 2010, 53(3): 659-676.

[27] WANG P, GUAN X, QIN T, *et al*. A data streaming method for monitoring host connection degrees of high-speed links[J]. IEEE Transactions on Information Forensics and Security, 2011, 6(3): 1086-1098.

[28] ZHAO Q, KUMAR A, WANG J, *et al*. Data streaming algorithms for efficient and accurate estimation of flow size distribution[A]. Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)[C]. New York, USA, 2004. 350-361.

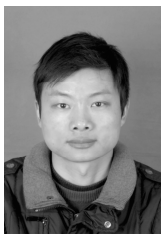
[29] KUMAR A, SUNG M, XU J, *et al*. A data streaming algorithm for estimating subpopulation flow size distribution[A]. Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)[C]. Banff, Alberta, Canada, 2005. 61-72.

[30] PAGH R, RODLER F F. Cuckoo hashing[J]. Journal of Algorithms, 2004, 51(2): 122-144.

[31] The CERNET traces dataset[EB/OL]. http://iptas.edu.cn/src/system.php, 2014.

[32] The CAIDA UCSD OC48 Internet traces dataset[EB/OL]. http://www.caida.org/data/passive/passive_oc48_dataset.xml, 2014.

作者简介：



周爱平（1982-），男，江苏泰州人，东南大学博士生，主要研究方向为网络测量、网络安全等。

程光（1973-），男，安徽黄山人，东南大学教授、博士生导师，主要研究方向为网络测量、网络安全、网络管理等。

郭晓军（1983-），男，山西长治人，东南大学博士生，主要研究方向为网络测量、网络安全等。

朱琛刚（1982-），男，江苏南京人，东南大学博士生，主要研究方向为网络测量、网络安全等。