Pervasive and Mobile Computing 48 (2018) 101–117

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

Fast flow volume estimation

Ran Ben Basat [a,*], Gil Einziger [b], Roy Friedman [a]

[a] Technion, Israel
[b] Nokia Bell Labs, Israel

ARTICLE INFO

Article history:
Available online xxxx

ABSTRACT

The increasing popularity of jumbo frames means growing variance in the size of packets transmitted in modern networks. Consequently, network monitoring tools must maintain explicit traffic volume statistics rather than settle for packet counting as before. We present constant time algorithms for volume estimations in streams and sliding windows, which are faster than previous work. Our solutions are formally analyzed and are extensively evaluated over multiple real-world packet traces as well as synthetic ones. For streams, we demonstrate a run-time improvement of up to 2.4X compared to the state of the art. On sliding windows, we exhibit a memory reduction of over 100X on all traces and an asymptotic runtime improvement to a constant. Finally, we apply our approach to hierarchical heavy hitters and achieve an empirical 2.4-7X speedup.

1. Introduction

Traffic measurement is vital for many network algorithms such as routing, load balancing, quality of service, caching and anomaly/intrusion detection [1–6]. Typically, networking devices handle millions of concurrent flows [7–9]. Often, monitoring applications track the most frequently appearing flows, known as *heavy hitters*, as their impact is most significant.

Most works on heavy hitters identification have focused on packet counting [10–13]. However, in recent years jumbo frames and large TCP packets are becoming increasingly popular and so the variability in packet sizes grows. Consequently, plain packet counting may no longer serve as a good approximation for bandwidth utilization. For example, in data collected by [14] in 2014, less than 1% of the packets account for over 25% of the total traffic. Here, packet count based heavy hitters algorithms might fail to identify some heavy hitter flows in terms of bandwidth consumption.

Hence, in this paper we explicitly address monitoring of flow *volume* rather than plain packet counting. Further, given the rapid line rates and the high volume of accumulating data, an aging mechanism such as a *sliding window* is essential for ensuring data freshness and the volume estimation's relevance. The window variant is motivated by load balancing applications. In these, one wishes to allocate resources to flows in a way that matches the consumed bandwidth. Volume estimation over sliding windows allows analysis of the most recent bandwidth consumption, which is a good indicator of the future. Hence, we study both streams and sliding windows.

Finally, per flow measurements are not enough for certain functionalities like anomaly detection and Distributed Denial of Service (DDoS) attack detection [15,16]. In such attacks, each attacking device only generates a small portion of the traffic and is not a heavy hitter. Yet, their combined traffic volume is overwhelming. *Hierarchical heavy hitters* (HHH) aggregates traffic from IP addresses that share some common prefix [17]. In a DDoS, when attacking devices share common IP prefixes, HHH can discover the attack. To that end, we consider volume based HHH detection as well.

* Corresponding author.
E-mail addresses: sran@cs.technion.ac.il (R. Ben Basat), gil.einziger@nokia.com (G. Einziger), roy@cs.technion.ac.il (R. Friedman).

https://doi.org/10.1016/j.pmcj.2018.06.002
1574-1192/

Before explaining our contribution, let us first motivate why packet counting solutions are not easily adaptable to volume estimation. Counter algorithms typically maintain a fixed set of counters [10,18–22] that is considerably smaller than the number of flows. Ideally, counters are allocated to the heavy hitters. When a packet from an unmonitored flow arrives, the corresponding flow is allocated the minimal counter [21] or a counter whose value has dropped below a dynamic threshold [20].

We refer to a stream in which each packet is associated with a *weight* as a *weighted* stream. Similarly, we refer to streams without weights, or when all packets receive the same weight, as *unweighted*. For unweighted streams, ordered data structures allow constant time updates and queries [10,21], since when a counter is incremented, its relative order among all counters changes by at most one. Unfortunately, maintaining the counters sorted after a counter increment in a weighted stream either requires to search for its new location, which incurs a logarithmic cost, or resorting to logarithmic time data structures like heaps. The reason is that if the counter is incremented by some value w, its relative position might change by up to w positions. This difficulty motivates our work.[1]

1.1. Contributions

We contribute to the following network traffic measurement problems: (i) stream heavy hitters, (ii) sliding window heavy hitters, (iii) stream hierarchical heavy hitters. Specifically, our first contribution is *Frequent items Algorithm with a Semi-structured Table* (FAST), a novel algorithm for monitoring flow volumes and finding heavy hitters. FAST processes elements in worst case $O(1)$ time using asymptotically optimal space. A major part of our contribution lies in the detailed formal analysis we perform, which proves the above properties, as well as in the accompanying performance study. We evaluate FAST on five real Internet packet traces from a data center and backbone networks. We demonstrate a speedup of up to a factor of 2.4X compared to previous works.

Our second contribution is *Windowed Frequent items Algorithm with a Semi-structured Table* (WFAST), a novel algorithm for monitoring flow volumes and finding heavy hitters in sliding windows. We evaluate WFAST on five Internet traces and show that its runtime is reasonably fast, and that it requires as little as 1% of the memory of previous work [23]. We analyze WFAST and show that it operates in constant time and is space optimal, which asymptotically improves both the runtime and the space consumption of previous work. We believe that such a dramatic improvement makes volume estimation over a sliding window practical!

Our third contribution is *Hierarchical Frequent items Algorithm with a Semi-structured Table* (HFAST), which finds hierarchical heavy hitters. HFAST is created by replacing the underlying HH algorithm in [24] (Space Saving) with FAST. We evaluate HFAST and show an asymptotic update time improvement as well as a 2.4-7X speedup on real Internet traces.

2. Related work

Our work addresses three related problems, which we survey below.

2.1. Streams

Probabilistic short counters, or *estimators*, represent large numbers using small counters by degrading precision [11,12,25]. By shrinking counters' size, more flows can be monitored in SRAM. But these methods still require maintaining a flow-to-counter mapping that often requires more space than the counters themselves. Sampling is also an attractive approach when space is scarce [26–28] despite the resulting sampling error.

Sketches such as *Count Sketch (CS)* [29] and *Count Min Sketch (CMS)* [30] are attractive as they enable counter sharing and need not maintain a flow to counter mapping for all flows. Sketches typically only provide a probabilistic estimation, and often do not store flow identifiers. Thus, they cannot find the heavy hitters, but only focus on the volume estimation problem. Advanced sketches, such as Counter Braids [31], Randomized Counter Sharing [32] and Counter Tree [33], improve accuracy but their queries require complicated decoding procedures that can only be done off-line.

In *counter based* algorithms, a flow table is maintained, but only a small number of flows are monitored. These algorithms differ from each other in the size and maintenance policy of the flow table, e.g., *Lossy Counting* [20] and its extensions [19], *Frequent* [22] and *Space Saving* [21]. Given ideal conditions, counter algorithms are considered superior to sketch based techniques. Particularly, Space Saving was empirically shown to be the most accurate [34–36]. Many counter based algorithms were developed by the databases community and are mostly suitable for software implementation. The work of [10] suggests a compact static memory implementation of Space Saving that may be more accessible for hardware design. Yet, software implementations are becoming increasingly relevant in networking as emerging technologies such as NFVs become popular.

Alas, most previous works rely on sorted data structures such as *Stream Summary* [21] or SAIL [10] that only operate in constant time for unweighted updates. As mentioned, existing sorted data structures cannot be maintained in constant time in the weighted updates case. Thus, a logarithmic time heap based implementation of Space Saving was suggested [35] for

[1] The most naive approach treats a packet of size w as w consecutive arrivals of the same packet in the unweighted case, resulting in linear update times, which is even worse.