Computer Communications 36 (2013) 927–938

Contents lists available at SciVerse ScienceDirect

Computer Communications

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

An efficient hybrid approach to per-flow state tracking for high-speed networks[☆]

Brad Whitehead[a], Chung-Horng Lung[a,*], Peter Rabinovitch[b]

[a] Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada
[b] Alcatel-Lucent, Ottawa, Canada

ARTICLE INFO

Article history:
Received 27 June 2012
Received in revised form 19 December 2012
Accepted 26 December 2012
Available online 16 January 2013

Keywords:
Network monitoring
Flow tracking
Bloom filter
High-speed networks
d-Left hashing

ABSTRACT

Maintaining per-flow information and state is a crucial topic in network monitoring. Tracking per-flow state is a relatively new area. Two main approaches have been proposed for tracking state: Binned Duration Flow Tracking (BDFT) and Fingerprint-Compressed Filter Approximate Concurrent State Machine (FCF ACSM). BDFT which uses Bloom filters is time efficient, whereas FCF ACSM using d-left hash tables has near-perfect memory efficiency but has higher computational cost. This paper presents a hybrid method (BDFT-H) by employing the best features of BDFT and FCF ACSM to achieve both time and space efficiency. Performance analysis and comparisons are conducted for BDFT, FCF ACSM, and BDFT-H. These methods are all intended for implementation on high-speed routers where resources such as memory and CPU time are limited. For the computational performance of the three schemes, we find that based on analysis, d-left hashing may require substantially more computational resources than Bloom filters. We also conduct simulations to compare the accuracy of these three schemes and the results show that all three methods can achieve over 99% accuracy on traces of real traffic. The proposed BDFT-H provides the best overall tradeoff between time and space efficiency. Both BDFT and FCF ACSM may have the false positive issue. This paper also presents two additional BDFT extensions: BDFT-FPR (false positive removal) and BDFT-FPC (false positive correction) to deal with the false positive issue. Performance comparisons for BDFT and these two BDFT extensions are also conducted using real traffic traces for comparison.

1. Introduction

Monitoring the transmission of various network protocols is crucial to ISPs, as better network control leads to better utilization which leads to lower costs. A common task to network measurement applications is to store some amount of state about individual flows. Monitoring per-flow state on high-speed routers requires an approach that is highly efficient in terms of both memory and processing capabilities. Packet sampling (e.g., 1 in 20 sampling) normally returns low accuracy. For long duration and high bandwidth flows, a naive implementation can manage to track some long duration flows. However, the memory usage could be very high, as the number of flows can be huge.

This paper proposes a method of tracking per-flow state that is both time and space efficient. The main idea is to integrate two efficient approaches; one is efficient in time, the other one is highly effective in memory usage but has higher computational cost. These two approaches are briefly summarized in the next two paragraphs. Section 3 discusses both approaches in details.

The first method, Binned Duration Flow Tracking (BDFT), was proposed [23,25] as a time efficient method for flow tracking. In brief, BDFT operates by grouping individual flows into "bins" which represent the current state of the flow. BDFT assigns time ranges to each state (bin) (e.g., 0–15 s, 15–45 s, 45–75 s, 75–105 s), and moves flows to the next time range (state) on a periodic basis. BDFT inherits much of its time and space efficiency from the use of counting Bloom filters [3] as the data structure which represents the bins.

The other approach is Fingerprint-Compressed Filter Approximate Concurrent State Machine (FCF ACSM, [3]) which can effectively track per-flow duration. FCF ACSM was mainly motivated by an observation obtained from the field: routers and network devices have to keep a large volume of the state of TCP connections and the application level QoS requirements. FCF ACSM was therefore proposed with an aim at improving memory usage. The paper also illustrated two potential applications of FCF ACSM to two real examples: queue management schemes for video flows and real-time control of P2P traffic.

☆ The project was sponsored by Ontario Centres of Excellence (OCE), and Alcatel-Lucent, Ottawa, Canada.

* Corresponding author. Tel.: +1 613 520 2600 x 2642.
E-mail addresses: bwhitehe@sce.carleton.ca (B. Whitehead), chlung@sce.carleton.ca (C.-H. Lung), peter.rabinovitch@gmail.com (P. Rabinovitch).

0140-3664/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comcom.2012.12.010

In short, FCF ACSM is highly memory-efficient [3] but has higher computational cost than BDFT. Both space and speed are crucial concerns to routers and network devices for an increasing trend that is more application aware services. We presented a hybrid of BDFT and FCF ACSM, which takes advantage of the best characteristics of each called BDFT-H [26]. This paper extends our initial effort by presenting the computational comparison of the three methods–BDFT, FCF ACSM, and BDFT-H in details for more general cases. It is both practically and theoretically important to consider more general cases for scalability analysis. Further, two extensions of [26], which deal with the false positive issue, are also evaluated and presented for comparison in this paper. Those two false positive specific extensions are BDFT-FPR (false positive removal) and BDFT-FPC (false positive correction).

Computational comparison is central to this paper. We adopt three main performance metrics for computational analysis: processing time, memory size used, and estimation accuracy. These metrics are important parts of the overall performance picture of each method. Processing time is directly related to the CPU resource. For the per-flow tracking problem, processing time is also strongly tied to the number of memory accesses for each packet or flow, and it has to be extremely efficient for router's forwarding rate [15]. In addition, accuracy for flow duration estimation is critical in determining the computational performance for per-flow tracking, as accuracy provides useful or necessary information for network utilization and management. The accuracy and memory usage of various methods is determined by running simulations with two real-world traffic traces, and the processing time is determined by evaluating critical CPU operations that are needed.

The main contribution of the paper is a hybrid method (BDFT-H) by adopting the best strengths of BDFT and FCF ACSM to achieve both time and space efficiency. For the computational performance of BDFT, FCF ACSM, and BDFT-H, we find that, based on analysis, d-left hashing may require substantially more computational resources than Bloom filters. We also conduct simulations using traffic traces to compare the accuracy of these three schemes and the results show that the proposed BDFT-H provides the best overall tradeoff between time and space efficiency.

The rest of the paper is organized as follows: Section 2 describes the background and recent state-of-the-art per-flow tracking methods. Section 3 discusses BDFT and FCF ACSM. Section 4 presents BDFT-H. Section 5 discusses computational analysis of these three methods. Section 6 presents the experiments and results for BDFT, FCF ACSM, BDFT-H, as well as two extensions of BDFT. Finally, Section 7 concludes our study.

2. Background

Tracking per-flow state is a relatively new area. Bloom filters and its variants such as counting Bloom filters [2,4] have become wide-spread in network monitoring. In [4], a number of applications of Bloom filters and its variants were discussed, such as distributed caching, P2P networks, packet routing, queue management, and network measurement infrastructure. The main reason is their ability to provide a time and space efficient data structure to represent a set of items when some errors are acceptable [2]. Some other Bloom filter variants are Space-Code Bloom filters [14] and Time-Decaying Bloom filters [9] which can track flow duration with medium accuracy. It is shown that a Bloom filter can be implemented in hardware and can scale to OC-192 (10 Gbps) speeds [1].

Bonomi et al. [3] presented two variations of a state tracking system using Bloom filters. Their first method uses a single counting Bloom filter to store a set of <flow, state> pairs. Their second approach uses counting Bloom filters to store both a count and a state in each cell corresponding to a flow's hashes. The main difference between their methods and ours is that our paper introduces the concept of using multiple bins (and therefore Bloom filters) to achieve high accuracy with low computational requirements.

Cuckoo hashing requires only a constant number of items to be moved for each insertion, depending on the load of the hash table. However, standard cuckoo hashing suits software applications, not high-speed routers [12]. In [13], the authors designed a scheme that allows at most one item to move during insertion, which results in higher space utilization. Both [12] and [13] consider the availability of content addressable memory that allows parallel lookups.

In [17], the authors proposed a near-optimal memory efficiency approach using counter braids. Multi-level counters are used in the approach to reduce *memory overhead*. However, the number of memory accesses per packet limits the computational efficiency.

Chen, et al. [8] presented an approach to tracking flow durations using two Bloom filters and sampling techniques. Their aim is to support anomaly detection and traffic engineering. Nevertheless, they only concern *traffic flows that have long durations*. Our approach can be used for all flows, including long duration flows.

Authors of [20] proposed the virtual vector data structure for network traffic monitoring and analysis. The main objectives of their approach are to study spread estimation, per-flow measurement, and long-duration flow detection. Their approach monitors every packet and stores summarized state information at a router, which is practically challenging to keep up with the line speed of modern routers [15,16].

Detection of long-duration flows is a primary goal for [8] and [20]. This goal can also be easily identified using BDFT, as the long-duration flows will be stored in the longer duration bins in BDFT or BDFT-H, see Section 3.1 for BDFT and Section 4 for BDFT-H, respectively.

Authors in [16] proposed a novel data structure called probabilistic multiplicity counting (PMC) to track per-flow probabilistically. The aim of PMC is to approximately keep track of *the number of packets per flow*. PMC is used for sampling data into only a single bit and extracting multiplicity from the single bit. The results show that PMC outperforms the approach presented in [14] and the standard sample technique. On the other hand, flow tracking is more than just keeping track of the number of packets per flow as used in this paper.

Li et al. [15] presented an approach for per-flow traffic measurement and analysis using a data encoding/decoding scheme for counter sharing among traffic flows. Their objective is to minimize *the number of memory access per packet* or computational cost on limited SRAM memory. It seems that the memory size required in their approach could be large, up to 8 MB for their experiments, compared with our approach (see Section 6 for experimental results).

Tracking traffic flows is also studied by Weaver et al. [22]. Instead of targeting general traffic monitoring and analysis, the authors investigated the characteristics of TCP Reset (RST) packets that are artificially injected into the network by ISPs or operators that cause the endpoints to shut down communication upon receipt.

Network monitoring involves storing a huge amount of data. An efficient storage technique was investigated by authors in [27]. The proposed technique performs data compression task for space reduction and provides the probability to execute operations directly on the compressed data instead of running decompression to reduce computational overhead. The technique, however, trades off accuracy for storage reduction. The controlled experimental results showed that the technique could achieve compression rate of 20% and approximate the original data within 5% relative error.

Further, network flow analysis is related to anomaly detection. Current anomaly detection approaches can be categorized into: