

Evolution of Cache Replacement Policies to Track Heavy-hitter Flows

Martin Zadnik
Brno University of Technology, Czech Republic
izadnik@fit.vutbr.cz

Marco Canini
EPFL, Switzerland
marco.canini@epfl.ch

Categories and Subject Descriptors

B.3.2 [Design Styles]: Cache memories

General Terms

Design

Keywords

cache, replacement policy, heavy-hitters, flows

1. INTRODUCTION

Flow-based network traffic processing, that is, processing packets based on some state information associated to the flows to which the packets belong, is a key enabler for a variety of network services and applications. This form of stateful traffic processing is used in modern switches [1] and routers that contain flow tables to implement forwarding, firewalls, NAT, QoS, and collect measurements.

Unlike Internet routing which scales sub-linearly with the number of connected hosts, flow-based traffic processing faces scaling challenges in that it potentially requires tracking and managing the state for each of millions of concurrent flows while keeping up with ever increasing data rates. In a number of cases, however, it is not necessary to track the state of each individual flow. Based on the generally known observation that a small number of flows account for a large amount of network traffic (e.g., see [2]), it has been suggested that scalable traffic measurement and accounting can be done by accurately measuring only the few large flows [3]. This can be generalized to other applications where the application goals can be met well enough by just focusing on the so called “heavy-hitters”. For example, a traffic shaping system may focus on rate-limiting of the large flows while the low-rate flows can utilize a small share of bandwidth at their will.

In the seminal work of Estan and Varghese [3], a memory-efficient structure called the Multistage filter has been introduced to define a scalable and efficient algorithm for iden-

tifying heavy-hitters. However, the limit of this approach is that a flow will only be accounted for once its volume has passed the filter and until this time no state can be assigned to that flow. As this limit is intrinsic to the filtering approach, the works that have extended the method above have inherited this limit. However, keeping the flow state since a flow’s first packet is critical for schemes that rely on the information carried in the first packet or first few packets. Methods for classifying traffic based on application identification (e.g., [4]) and network security schemes (e.g., [5]) are important classes of applications that require information from the first few packets.

In this work, we propose a different approach that overcomes this limitation. Specifically, we treat the problem of *identifying and tracking* heavy-hitters as that of finding a cache replacement policy that strives to avoid evicting the heavy-hitters from the flow table (from now flow cache). The intuition is that, if in the presence of a full cache and a new flow starting (causing a cache miss) the policy only chooses to evict flows that are not heavy-hitters (or unlikely), then the state of heavy-hitters is definitely preserved in the cache. Of course, we assume that the cache size has to be smaller than the total number of concurrent flows due to memory costs and therefore the need for a replacement policy.

Our goal, granted that the flow cache is larger than the number of all concurrently active heavy-hitters, is to find a replacement policy (RP) that has the least number of evicted heavy-hitters or using caching terminology minimizes the miss rate for heavy-hitters. We use the number of heavy-hitters that witness a cache miss as a metric to capture the effectiveness of a RP – the objective is to reduce this number. Finding such a RP can be difficult because the access pattern is influenced by a large number of factors including flow size distribution, flow rate, and other traffic dynamics. We propose using Genetic Algorithm (GA) to explore the space of possible RPs to identify the most effective.

2. DESIGN

The role of a replacement policy is to reorder flow states based on their access pattern. Each packet causes one cache access and one execution of the RP. If the current packet causes a cache miss (i.e., a new flow arrives) and the cache is full then the flow at the end of the list is evicted. Formally, we can express a RP as a pair $\langle s, U \rangle$ where s is a scalar representing the zero-based position in the list where new flow states are inserted and U is a vector (u_1, u_2, \dots, u_N) which defines how the flows are reordered. Specifically, when a flow F stored at position $pos_t(F)$ is accessed at time t ,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ANCS’10, October 25-26, 2010, La Jolla, CA, USA.

Copyright 2010 ACM 978-1-4503-0379-8/10/10 ...\$10.00.

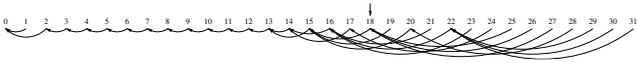


Figure 1: An example of RP produced by GA using the Caida dataset.

$RP = \langle 18, (0, 0, 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 13, 14, 14, 15, 15, 15, 16, 16, 16, 17, 18, 18, 18, 20, 22, 22, 22) \rangle$.

its new position is chosen as $pos_{t+1}(F) = u_{pos_t(F)}$, while all flows stored in between $pos_{t+1}(F)$ and $pos_t(F)$ see their position increased by one. As an example, representation of the LRU policy for a cache of size 4 is $LRU = \langle 0, (0, 0, 0, 0) \rangle$.

The vector-based definition of a RP is a well fit to encode the representation of a candidate solution for Genetic Algorithm. It supports the standard genetic operators for mutation and crossover. Mutation modifies a particular value in the vector with given probability p_{mut} while crossover swaps parts of the vector between two solutions with probability p_{cross} . The RP evolution is performed offline using traces from real network traffic.

Depending on the application, the definition of a flow changes adequately. One that is commonly used identifies a flow based on the 5-tuple composed of its IP addresses, port numbers and protocol. In our work, we consider a flow to be a unidirectional stream of packets sharing the same 5-tuple, but our approach can be easily generalized to allow the flow identifier to be a function of the header field values. We use a 60s timeout to determine the end of a flow unless we observe the TCP connection tear down.

We define a *heavy-hitter* as a flow that utilizes more than a certain percentage of a link bandwidth during its whole lifetime. In order to avoid bias from short-lived flows which overall do not carry significant amount of traffic (verified by measurements on several traffic traces), we require a heavy-hitter to exist for at least five seconds. Therefore, we compute a flow’s link utilization as $\frac{totalbytes}{max(5, lifetime)}$. Throughout this paper, we group flows into three reference categories based on their utilization: very large flows ($> 0.1\%$ of the link capacity), large flows (between 0.1% and 0.01%), medium flows (between 0.01% and 0.001%). We then report how well our approach perform for each category.

3. EVALUATION

We use traces of different Internet backbone traffic, however the one used in this paper is an anonymized, unidirectional 15-min trace from the Caida archive collected at the 10 Gbps Equinix San Jose link (dirA on July 17th 2008 at 13:00 UTC, 10 Gbps link, avg/min/max active flows: 1.7M/179K/1.8M) [6].

Table 1 summarizes the results obtained by evaluation of engineered policies as well as results of policies evolved by GA. S3LRU-IN7 means Single Step Segmented LRU [7] in which new flow states are inserted at position seven, SLRU-IN21 is Segmented LRU [8] with insert position 21. The policy evolved by GA is called GARP. and the evolved replacement vector is displayed on Fig. 1.

The experiments show that the GARP performs two times better to well-known RPs in keeping the state of heavy-hitters seamlessly.

Unfortunately, modification of replacement policy on current platforms is difficult if possible at all. On the other hand FPGA provides memory as well as logic to implement cache

RP	Cache miss		
	$\geq 0.1\%$	(0.1%, 0.01%)	(0.01, 0.001%)
LRU	0%	23%	23%
SLRU-IN21	0%	19%	15%
S3LRU-IN7	0%	15%	18%
GARP	0%	8%	9%
OPT	0%	0%	0%

Table 1: Cache miss for heavy-hitters under several RP (Cache size = 128K, data set: Equinix). OPT does not induce any cache miss because the cache is large enough to keep all heavy-hitters.

of arbitrary structure. Based on our previous experience [9] we propose 32 set associative cache composed of BlockRams, comparators and multiplexors capable of accommodating any RP being evolved by a previous scheme. We intend to build this cache to carry out experiments with various applications such as OpenFlow, NetFlow, traffic shaping, route caching.

Acknowledgment

This work was partially supported by the BUT FIT grant FIT-10-S-1 and the research plan MSM0021630528.

4. REFERENCES

- [1] N. McKeown et al. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, 2008.
- [2] A. Feldmann et al. Deriving traffic demands for operational ip networks: methodology and experience. *IEEE/ACM Trans. Netw.*, 9(3):265–280, 2001.
- [3] C. Estan and G. Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3):270–313, 2003.
- [4] W. Li et al. Efficient application identification and the temporal and spatial stability of classification schema. *Comput. Netw.*, 53(6):790–809, 2009.
- [5] G. Zhao, J. Yang, G. S. Hura, L. Ni, and S.-H. S. Huang. Correlating TCP/IP Interactive Sessions with Correlation Coefficient to Detect Stepping-Stone Intrusion. *Advanced Information Networking and Applications, International Conference on*, 0:546–551, 2009.
- [6] The caida anonymized 2008 internet traces. http://www.caida.org/data/passive/passive_2008_dataset.xml.
- [7] M. Zadnik et al. Tracking elephant flows in internet backbone traffic with an fpga-based cache. In *19th International Conference on Field Programmable Logic and Applications*, pages 640–644, 2009.
- [8] R. K. et al. Caching strategies to improve disk system performance. *Computer*, 27(3):38–46, 1994.
- [9] M. Canini et al. Experience with high-speed automated application-identification for network-management. In *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '09)*, pages 209–218, 2009.