# ALFE: A Replacement Policy to Cache Elephant Flows in the Presence of Mice Flooding

Tian Pan, Xiaoyu Guo, Chenhui Zhang, Wei Meng and Bin Liu†
Tsinghua National Laboratory for Information Science and Technology
Department of Computer Science and Technology, Tsinghua University, Beijing 10084, China

***Abstract*—Flow-based packet processing exists widely in a variety of network applications, where a large sized flow table is built to keep the alive flow records. To accelerate the search speed of the flow table, numerous systems employ cache mechanism to track the most recently referenced flows. However, network traffic exhibits some different characteristics from the workload of the general computational tasks, and classic replacement policies like LRU, Random, fail to perform well in the network scenarios. To develop a network-oriented flow cache replacement policy, we propose ALFE (Adaptive Least Frequently Evicted) based on the observations of traffic's heavy tailed feature and the statistically positive correlation between the flow size and the flow cache evict times. Specifically, the correlation helps us identify elephant flows at a tiny extra cost of a few more bits allocated to each flow entry. For those who are identified as possible elephant flows, ALFE favors their priorities in the cache, thus preventing them from being flooded by the massive mice flows. A prototype system employing ALFE policy is elaborately designed and implemented besides extensive simulations. Experimental results indicate that with 1K cache entries, ALFE can achieve up to 15% higher cache hit rate than LRU on real traces.**

## I. Introduction

Flow-based packet processing exists widely in a variety of network systems and applications including firewalls, QoS, NAT, NetFlow, OpenFlow, etc., where a flow table is usually adopted to record each flow's states as well as the corresponding routing policies. The flow table is queried by every incoming packet associated to different flows.

Since network link bandwidth keeps increasing rapidly in recent years, on-chip SRAM is insufficient to store the ever-increased large table due to its limited size and the huge number of concurrent flows (e.g. the concurrent flow number of an OC-192 link is around millions [1] and each flow record's size will mostly reach up to one hundred bits, or even more). Therefore, a full flow table has to be kept in the large sized but slow off-chip DRAM, which suffers from a speed bottleneck. Due to the discovery of the temporal locality in transport layer traffic [2], numerous flow-based systems employ flow cache located in fast SRAM to hold the frequently accessed flow records to speed up the processing [3], [4], [5], thus designing a cache replacement policy has become a crucial issue to improve flow-based system's performance.

†Corresponding author: liub@tsinghua.edu.cn.
Others: {pant09, guoxiaoyu10}@mails.tsinghua.edu.cn, {asterix314, francismw08}@gmail.com.

Supported by NSFC (61073171, 60873250), Tsinghua University Initiative Scientific Research Program, the Specialized Research Fund for the Doctoral Program of Higher Education of China(20100002110051).

To reduce the implementation complexity and cost, some systems use classic replacement policies, e.g. LRU [3], Random [4]. However, unlike program instruction and data reference patterns of the general purpose processor, network traffic exhibits different characteristics. Recent literature indicates that there exists some heavy-tailedness in network traffic that a small number of flows, namely, elephant flows, account for a large percentage of traffic volume [6]. For this reason, LRU policy is argued for its poor replacement performance to evict the most frequently accessed flows during the burst of mice flows possibly [5], [7]. Identically, Random policy also shows no utilization of the above traffic patterns. To alleviate the effect of mice flooding, segment-based replacement policies (e.g. SLRU [8], $S^3$-LRU [5]) divide the cache set into segments with distinct replacement priorities, attempting to hold the most active flows in good cache entries. The main drawback of these policies lies in that the cache itself does not have the pre-knowledge to tell the elephant flows (in these policies, flows are always assigned higher replacement priority in a temporal time window if recently referenced, regardless of information on referenced frequency in history). The experimental results show that $S^3$-LRU is only slightly better than LRU when traffic load becomes heavy with more concurrent flows [7].

When designing a flow cache replacement policy, we need take the following factors into consideration: 1) the network traffic characteristics, e.g. heavy-tailed distribution, should be effectively utilized; 2) the replacement policy should be better designed to adaptively cope with the dynamic changes in network links; 3) the flow cache structure should not be too complicated to implement. To meet these requirements, in this paper, we design a flow cache replacement policy called Adaptive Least Frequently Evicted (ALFE). The proposed policy pays a few extra bits to each flow entry to record the cache evict times, acting as an indicator of the flow's attribute: either an elephant flow or a mouse one. Once the flow's attribute is decided, ALFE treats different classes of flows more on purpose by assigning elephant flows a higher priority when staying in the cache, preventing them from being possibly flooded by massive mice flows. The priority threshold parameters are designed adjustable and adaptive to the changing network conditions. Compared with other cache replacement policies, ALFE can achieve a higher cache hit rate.

**Paper organization:** Section II studies the OC-192 traffic characteristics. Section III details the ALFE policy. Section IV

978-1-4577-2053-6/12/$31.00 ©2012 IEEE

provides a prototype design of flow cache structure with ALFE. We evaluate ALFE's performance in Section V and conclude the paper in Section VI finally.

## II. Traffic Characteristics

In this section, we provide a data mining on OC-192 traffic characteristics. The studied trace is obtained from CAIDA [1] of 2.2M flows and 60 seconds duration.

### A. Traffic Workload Profiling

Figure 1 depicts the flow/packet number distribution on the flow length of packets, which again affirms the heavy-tailedness of network traffic. Though not taking advantage of flow number, the elephant flows account for most of the packets, which also mean a high proportion of total traffic volume (e.g. 5.57% of the flows whose length of packets is larger than 20, account for nearly 72.7% of the total packets). However, those classic replacement policies, like LRU, learn less about this hint. Since the flow cache is organized in the granularity of flows rather than packets, though elephant flows weigh heavily in packet numbers and traffic volume, they do not gain any advantage over mice flows when competing for the limited cache entries due to their smaller flow numbers, thus more likely to be evicted when burst of mice occurs.

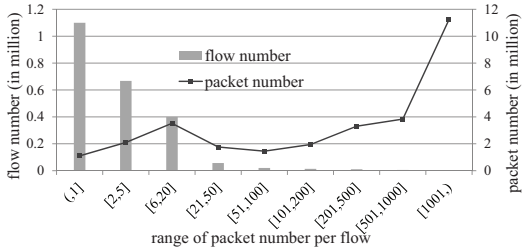


Fig. 1. Heavy-tailed distribution of OC-192 traffic

### B. Cache Trashing under LRU Policy

In Figure 2, using LRU with 1K cache entries in 8-way set associativity, elephant flows (here we classify flows containing more than 20 packets into elephant flows, and less than 4 packets into mice flows) of most traffic volume suffer from more cache evict times than mice flows. Specifically, mice flows suffer from a relatively low evict times (0-3 times) due to their small size (the value of evict times is always smaller than flow's length of packets). Though, there is some overlap between "elephant area" and "mid-size area", most flows can be classified into one of the three groups correctly according to their cache evict times.

To evaluate the correlation between the flow length of packets and the cache evict times quantitatively, we calculate their coefficient according to

$$corr(X_l, X_e) = \frac{E[(X_l - \mu_{X_l})(X_e - \mu_{X_e})]}{\sigma_{X_l}\sigma_{X_e}} \quad (1)$$

Where $\sigma$ is the standard deviation, $E$ is the expected value operator and $\mu$ is the expected value, $X_l$, $X_e$ refer to random variables of the flow length and the evict times respectively.

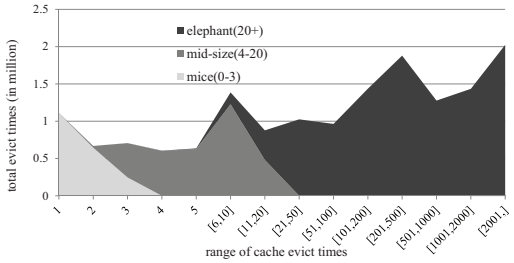


Fig. 2. Cache trashing under LRU replacement policy

$corr_{LRU}$ is around 0.71 and $corr_{RANDOM}$ is around 0.80 in our experiment, which indicate there is a positive correlation between the cache evict times and the flow length.

### C. Traffic Burstiness

Although Figure 1 shows the heavy-tailedness of the OC-192 traffic, the cache hit rate may still be very high if most packets of the same flows arrive at a burst (i.e. high locality). To quantify this, we define evict distance $D$ for a particular cache reference to be the total number of cache evicts which have occurred since the last reference of the same flow in a cache set. E.g. the reference stream in a particular cache set is $\{1, 3(e), 2(e), 4, 2, 7(e), \dot{1}\}$ ("e" represents for an evict), and the current cache reference is $\dot{1}$. Thus $D$ for $\dot{1}$ is 3. For a particular flow under LRU, if $D$ is always smaller than cache's set associativity, the flow will not suffer from a cache miss. In

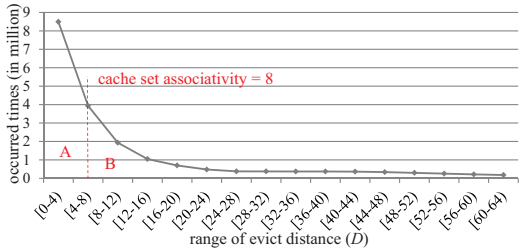


Fig. 3. Occurred frequency distribution on evict distance $D$ (elephant flows)

Figure 3, under the configuration of 1K cache entries with 8-way set associativity, evict distance $D$ of elephant flows is profiled under LRU policy. The evict distance distribution reflects the traffic burstiness (larger area of zone A means better traffic burstiness). As the value of $D$ is less than the set associativity, packets in zone A will not suffer from a cache miss, while all packets in zone B will suffer from cache misses. If we provide elephant flows with a higher priority in the cache replacement process, which equally means to decrease the value of $D$, more packets will migrate from zone B to zone A, thus increasing the cache hit rate of elephant flows.

## III. ALFE Cache Replacement Policy

### A. Basic Idea

The observation from II-A gives us a clue that to increase the cache hit rate, we'd better track the small number, but large traffic volume elephant flows instead of massive mice flows. The observation from II-B implies a relationship between

elephant flows and cache evict times. These above two hints provide us an opportunity to improve the cache performance.

The flow cache is indexed with part of hash value from 5-tuple. A few extra bits per flow entry are allocated as the counter to record how many times a flow record is swapped from the cache, which differentiates the possible elephant flows from the mice flows. Once a possible elephant flow is identified, higher priority to stay in the cache is assigned via the segment-based cache set partition (Figure 4). In ALFE, the

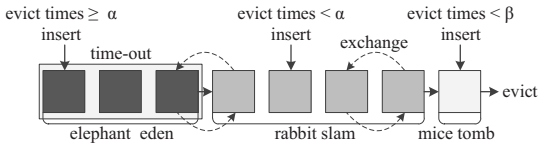


Fig. 4. ALFE cache set structure

cache set is divided into three segments of different sizes and replacement strategies for different flow lengths of packets. Flow records are inserted into one of the three segments according to their evict times when referenced. As the value of the evict times is always smaller than flow's length of packets, real mice flows will hardly pollute the rabbit slam and the elephant eden. E.g. if $\beta$ is defined as mice flow's upper threshold of length of packets, as well as the boundary of the evict times between the mice tomb and the rabbit slam, those mice flow records (packet number $< \beta$) will not be inserted into elsewhere other than the mice tomb. Based on the case study in Section II, flows of 1-5 packets account for nearly 80% of the total flow numbers, but less than 20% of the traffic volume. If we let $\beta$ equal to 5, 80% mice will not flood into the limited flow entries to squeeze elephants and rabbits, at the expense of only 20% sacrificed short flow traffic. Thus without the disturbance from the mice flows, the flow records in the elephant eden are assigned higher priority in the replacement process for staying longer in the cache (with smaller evict distance $D$). The referenced records located in the cache set will be exchanged one step forward with its precursor. Considering that some flows arrive at a burst, this replacement strategy for dealing with cache hit opens up another path for large bursty flows to enter the elephant eden without repeated cache swapping, thus becoming a complement to the historically evict times based strategy. The flow records in the mice tomb and the rabbit slam are edged out by newly inserted ones while flow records in the elephant eden are designed to be timed-out directly when not referenced for a long time. Once evicted, the evict counter is incremented by 1. Those large bursty flows once entered elephant eden are favored by increasing their evict counters by a larger number. Time-outed elephants leave "hole"s in the elephant eden where newly inserted ones will occupy. A range of time-out values can be applied to offer the elephant flows different levels of priorities. The width of the evict counter should be limited to avoid becoming large memory overhead to the flow record. In the experiment, we allocate 5-bit evict counter per flow and just "freeze" the counter to all 1s if it becomes overflowed.

### B. Algorithm

Algorithm 1 formalizes the replacement procedure when flow records are referenced either in the DRAM or in the flow cache, together with the asynchronous time-out process.

**Algorithm 1** Replacement(cache set $cs$, flow record $fr$)

```
1: if fr.src == DRAM then
2:   // cache miss, flow swapped from DRAM
3:   if fr.evict_times < β then
4:     insert_rotate(fr, cs[MICE]);
5:   else if fr.evict_times < α then
6:     insert_rotate(fr, cs[RBT_MID]);
7:   else
8:     if cs.elephant.hole.exist == 1 then
9:       fr.loc ← cs.elephant.hole.loc; // fill the hole
10:    else
11:      // insert into the head entry of the elephant eden
12:      insert_rotate(fr, cs[ELE_HEAD]);
13:    end if
14:  end if
15:  // favor bursty large flows when evicted
16:  !cs.evicted.bst_bit ? cs.evicted.evict_times += 1 :
     cs.evicted.evict_times += MORE_TIMES;
17: else
18:  // cache hit
19:  if fr ∈ ELEPHANT_EDEN then
20:    fr.ref_bit ← 1; // set the reference bit
21:    // move to the head of elephant eden and rotate others
22:    rotate(fr, cs[ELE_HEAD]);
23:  else
24:    if fr.loc == RBT_HEAD then
25:      // enter the elephant eden
26:      fr.ref_bit ← 1; fr.bst_bit ← 1;
27:      if cs.elephant.hole.exist == 1 then
28:        fr.loc ← cs.elephant.hole.loc;
29:      else
30:        // exchange with precursor
31:        exchg(fr, cs[fr.loc + 1]);
32:      end if
33:    else
34:      exchg(fr, cs[fr.loc + 1]);
35:    end if
36:  end if
37: end if
38: // asynchronous time-out (using second chance)
39: if time-out gets fulfilled then
40:  for all fr ∈ ELEPHANT_EDEN do
41:    if fr.ref_bit == 1 then
42:      fr.ref_bit ← fr.ref_bit − 1; // second chance
43:    else
44:      evict(fr); // if cleared, evict it
45:    end if
46:  end for
47: end if
```

### C. Adaptive Parameters

*1) time-out value:* The setting of the time-out value reflects the replacement priority assigned to the elephant flows. If the time-out value is too high, which means elephant flows stay in the cache for too long a time and squeeze the rabbits and mice; if it is too low, elephant flows is not given the appropriate priority, resulting in a larger elephant flow's cache miss rate. In the elephant eden, elephant flows will not be disturbed by rabbit and mice flows. In some degree, the physical meaning of the time-out value in ALFE equals to the evict distance (II-C) in LRU if large bursty flows are not considered. According to Figure 3, if we set the time-out value to be the time during which eight evicts occurred in the cache set, the packets in area A will not suffer from cache misses approximately. If we increase the time-out value to be the time of more evicts, some packets in area B will migrate to area A, which means a higher priority for elephant flows. Note that we use second chance time-out strategy, which means the actual time-out is 1.5x above discussed time-out value if the duration of packet intervals in a flow follows a uniform distribution. Therefore, the time-out value in second chance strategy can be expressed as $T = \frac{1}{1+\frac{1}{2}} * \sigma * t(associativity)$. $t(x)$ is the time duration of $x$ evicts occurred in the cache set. As the time-out value becomes larger, the packet number in area B gets smaller. Here we set $\sigma$ to be 2, achieving a 2x evict distance time-out of the cache set associativity.

*2) $\alpha$:* If $\alpha$ is too high, fixed-sized elephant eden will be "holey" and not best utilized; otherwise, elephants and rabbits may be treated equally. The best case is when the newly arrived elephants just fill the "hole"s left by the recently evicted old ones. But it is hard to guarantee and we leave some redundant $\delta$ to be the times of $rotate()$ and $insert\_rotate()$ operations to the elephant eden in a time-out duration. If there is a "hole" when sampled, we lower $\alpha$; otherwise, if $\delta$ is too high, we increase $\alpha$. In our experiment $\delta$ is set to ensure that the times of $rotate()$ and $insert\_rotate()$ operations to the elephant eden is $\frac{1}{10}$ of that to the mice tomb and rabbit slam.

*3) $\beta$:* $\beta$ blocks massive mice flows in the mice tomb. According to Figure 1, short flows occupy a large percentage of the total flow number. So do the flows' evict times. Therefore we can use sampling to obtain the appropriate value of $\beta$ most probably as: $\beta_i = a * \beta_{i-1} + b * \beta_i^{'}$, $(a + b = 1)$; $\beta_0^{'} = Avg(Random\_Sample(fr.evict\_times)) + \zeta$. $Random\_Sample()$ is the procedure to select a subset of flows' evict times randomly; $Avg()$ is the average procedure; $a$ is the historical weight factor to smooth the glitches during the sampling; $\zeta$ is the redundant factor (evict times is always smaller than the flow length of packets). In the experiment, we sample 100 flow records and set $a = \frac{1}{2}$ and $\zeta = 1$.

## IV. ALFE Cache Implementation

### A. Cache Unit

The structure of a single cache unit is depicted in Figure 5. We use the dual-port RAM block to construct such a unit, which is quite similar with the content addressable memory (CAM), but can execute write/read on the same memory entry in one clock cycle. To prototype the cache unit for experimental purpose, we use TriMatrix memory block in Altera's Stratix FPGA. The unit operates on a pipeline of 2 stages, exporting a result per clock cycle.

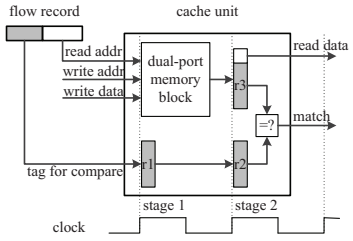


Fig. 5. Flow cache unit

In the first stage, the dual-port memory completes one read and one write operation simultaneously. The read address is hashed from 5-tuple flow-ID, which is also loaded into r1 as a flow tag for later comparison. In the second stage, the read output of memory consisting of the (flow-ID, policy) pair is loaded into r3 while the content of r1 is transferred down the pipeline into r2. At this point, the comparator is able to produce a single bit "match" signal indicating if the tag from the flow record matches the part of memory output. If yes, the read data will output the search result.

### B. Cache Cascading with ALFE Control Logic

Figure 6 shows the cascading of identical cache units. The inter-connections together with a combinational control unit implement the ALFE replacement policy. The cache control will produce the "SELECT" bus signal via the processing of the inputs such as "RDDATA". The cache control will also manage the multiplexers in between the cache units. The multiplexers in turn will control where the data put into the cache unit comes from. The combination of several "SELECT" bits implements the data movement/replacement in the cache set. E.g. in Figure 6, multiplexer M2 passes the data into cache unit U2 from INSERT (data swapped from DRAM), U1 (cache set rotation when replacement is happened) or U3 (U3 is referenced and promoted into higher location of the cache set) according to the encoding of "SELECT" signal.

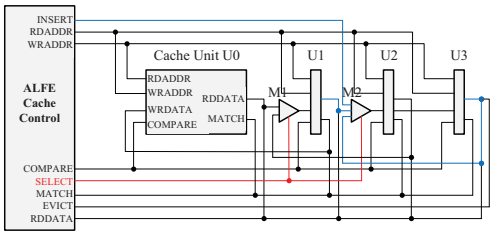


Fig. 6. Cache cascading with ALFE control logic

### C. Implementation Considerations

*1) Where to Attach the Evict Counter:* The on-chip flow cache has a limited size. Attaching the per flow evict counters to it will decrease the cache entry number if the total cache size is a constant, thus lowering the cache hit rate on the contrary. As the evict counter is only referenced for selecting the insert

location when the flow records in DRAM are swapped into the cache, we attach the per flow evict counters to DRAM rather than to cache for saving on-chip resource.

*2) When to Write the Evict Counter:* It is easy to think of the way to increment the evict counter in DRAM when the corresponding flow record is swapped from the cache. But the memory write will additionally consume DRAM's bandwidth. In a cache/DRAM structure, the DRAM path will mostly be the performance bottleneck of the entire system, and the frequency of writing the evict counter is as large as the cache miss rate, which can not be ignored. To eliminate the additional cost of memory bandwidth, we combine the evict counter write with the time-out labeling operation when the flow query suffers from a cache miss and queries DRAM. As a current cache miss corresponds to a previous cache evict of the same flow, we can use cache miss times to substitute for evict times equally while reducing memory bandwidth requirement.

## V. Evaluation

### A. Comparison of Cache Hit Rate

Table I shows cache hit rates obtained using different replacement policies with different cache entry numbers (the cache set associativity is 8). ALFE outperforms Random, LRU, SLRU, $S^3$-LRU and reaches up to 15% higher cache hit rate than LRU on real OC-192 traces.

TABLE I
Comparison of Cache Hit Rates

| | 0.5K | 1K | 2K | 8K |
|---|---|---|---|---|
| *Random* | 0.402 | 0.453 | 0.541 | 0.683 |
| *LRU* | 0.425 | 0.472 | 0.573 | 0.711 |
| *SLRU* | 0.468 | 0.511 | 0.600 | 0.762 |
| *$S^3$-LRU* | 0.470 | 0.520 | 0.609 | 0.774 |
| *ALFE* | 0.489 | 0.561 | 0.637 | 0.801 |

### B. Migration

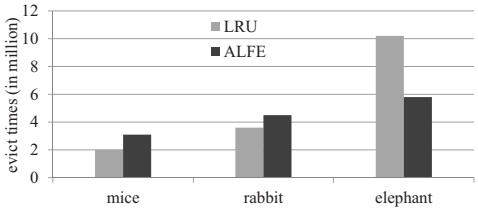


Fig. 7. Evict times migration within 3 categories of flows

ALFE assigns a higher priority to the elephant flows, which will inevitably degrade the cache hit rate of the mice and rabbit flows. As the elephant flows occupy a higher percentage of the total traffic (heavy-tailedness), in ALFE, we obtain more cache hit times from elephant flows than which we lose in mice and rabbit flows. Figure 7 demonstrates the migration of evict times within 3 categories of flows from LRU to ALFE.

### C. Time-out

Figure 8 shows the relationship between the time-out value and the evict times. The evict times of elephants are getting smaller as the timeout value increases. As Figure 3 shows, most evicts occur in the range from [0-4) to [16-20), thus when time-out value is beyond some degree, the decrease of elephant evict times will be limited. But at this time, the evict times of rabbits and mice will be quite large. In the experiment, we reach a minimal total cache evict times (ie. maximum cache hit rate) when time-out value is set to be 2x evict distance of the cache set associativity.

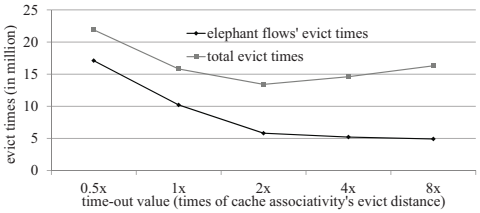


Fig. 8. Time-out (times of cache associativity's evict distance) vs evict times

### D. Algorithm Limitation

There are two prerequisites to effectively use ALFE: one is traffic's heavy-tailedness; the other is the relatively low burstyness. Backbone traffic with large concurrent number of flows may often have the above features. Otherwise, ALFE will degrade to LRU or even worse for its cache control logic overhead, which will decrease the total cache entry number.

## VI. Conclusion

A good cache replacement policy applied to the network applications for flow-based system is important to improve the system's performance. Given the existing algorithms fail to achieve high hit rate, we have proposed a new cache replacement policy named Adaptive Least Frequently Evicted to track elephant flows in the presence of mice flooding. ALFE utilizes the traffic characteristics of heavy-tailedness as well as the positive correlation between the elephant flows and the cache evict times to accurately identify the elephant flows, and then gives elephant flows higher priorities in the cache, preventing them from being flooded by a large number of mice flows. Our experimental results show ALFE can increase the cache hit rate remarkably compared with the existing policies.

## References


[1] C. Walsworth, E. Aben, kc claffy, and D. Andersen, "The caida anonymized 2010 internet traces," http://www.caida.org/data/passive/passive_2010_dataset.xml.

[2] K. Thompson, G. Miller, and R. Wilder, "Wide-area internet traffic patterns and characteristics," *Network, IEEE*, vol. 11, no. 6, pp. 10 –23, nov/dec 1997.

[3] D. Schuehler, J. Moscola, and J. Lockwood, "Architecture for a hardware-based, tcp/ip content-processing system," *Micro, IEEE*, vol. 24, no. 1, pp. 62 – 69, jan.-feb. 2004.

[4] J. M. Gonzalez, V. Paxson, and N. Weaver, "Shunting: a hardware/software architecture for flexible, high-performance network intrusion prevention," in *Proceedings of the 14th ACM conference on Computer and communications security*, ser. CCS '07. New York, NY, USA: ACM, 2007, pp. 139–149.

[5] M. Canini, W. Li, M. Zadnik, and A. W. Moore, "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*, ser. ANCS '09. New York, NY, USA: ACM, 2009, pp. 209–218.

[6] L. Quan and J. Heidemann, "On the characteristics and reasons of long-lived internet flows," in *Proceedings of the 10th annual conference on Internet measurement*. ACM, 2010, pp. 444–450.

[7] M. Zdnk and M. Canini, *Evolution of Cache Replacement Policies to Track Heavy-Hitter Flows*, 2011.

[8] R. Karedla, J. Love, and B. Wherry, "Caching strategies to improve disk system performance," *Computer*, vol. 27, no. 3, pp. 38 –46, mar 1994.