Fair Service for Mice in the Presence of Elephants

Seth Voorhies [a], Hyunyoung Lee [a], Andreas Klappenecker [b,*]

[a] *Department of Computer Science, University of Denver, Denver, CO 80208, USA*

[b] *Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, USA*

Abstract

We show how randomized caches can be used in resource-poor partial-state routers to provide a fair share of bandwidth to short-lived flows that are known as mice when long-lived flows known as elephants are present.

Key words: algorithms, randomized algorithms, caching, probabilistic queues.

1 Introduction

Offering internet access in apartments and hotels is now commonplace. However, it is not untypical in such networks that a few users watching streaming video broadcasts consume most of the available bandwidth, while – at the same time – users browsing the web experience unreasonably long delays. In case of congestion, a router may be forced to drop packets. One can observe from experimental data that long-lived flows (the elephants) receive a larger share of the bandwidth than short-lived flows (the mice) in many common queuing policies such as drop tail, fair queuing, random early detection, stochastic fair queuing, and class based queuing.

* Corresponding author.
Email addresses: hlee@cs.du.edu (Hyunyoung Lee), klappi@cs.tamu.edu (Andreas Klappenecker).

Recently, the elephant flow phenomenon gained much interest in the networking community, see, for example [1]. An inexpensive, scalable solution to monitor elephant flows in a router without keeping information about all flows was proposed by Smitha, Kim, and Reddy [2]. The basic idea of their scheme is (a) to sample the packets and insert with a certain probability their flow identifiers into a cache; (b) keep count of the number of packets per flow for all flows in the cache; (c) purge packets of elephant flows in the cache from the queue if necessary. We propose a variation of this scheme that does not use step (b) but still achieves the same goal. We give a thorough analysis and prove that our randomized caching scheme is able to identify elephant flows with high probability.

2 The Algorithm

We propose a queuing mechanism for a router that uses a randomized cache with least-recently-used eviction strategy for congestion control. The router maintains a queue Q of maximal size q and a cache T of size t. An incoming packet m is enqueued in a queue Q of pending packets and a flow identifier $\mathrm{id}(m)$ is calculated from the header information of the packet. The flow identifier $\mathrm{id}(m)$ is inserted with probability α into the cache T. If the queue Q of the router exceeds the maximum number q of allowed packets due to network congestion, then all packets m with packet identifier $\mathrm{id}(m)$ in T are deleted from the queue Q. The pseudocode for this queuing algorithm is shown in Algorithm 1.

Algorithm 1 Probabilistic Queuing.

ProbabilisticQueue(α, q)
/* α is cache insertion probability, q specifies the maximum length of the queue Q */
Initialization:
1: $pending := 0$; $T :=$ empty list; $Q :=$ empty queue;
When a request item m arrives:
2: $pending := pending + 1$;
3: **if** ($pending > q$) **then** /* congestion, need to dequeue requests in Q */
4: **if** (T is empty) **then**
5: choose at most t differing ids among the most recent requests and put these ids in T;
6: **forall** id in T **do** Q.dequeue(id);
7: $pending := pending -$ (the number of dequeued requests);
8: Q.enqueue(m);
9: RandomCaching(id(m),α);

The operation Q.enqueue(m) inserts a packet m at the tail of Q. The operation Q.dequeue(id)

removes *every* packet with flow identifier id from Q. The last statement calls a randomized version of an LRU cache; the corresponding pseudocode is shown in Algorithm 2.

Algorithm 2 Randomized Cache.

```
RandomCaching(id, α)
1:  if (random_put(α) = true) then /* id will be inserted into T with probability α */
2:      if (id exists in T) then
3:          move id to the top of T;
4:      else /* id does not exist in T */
5:          if (T contains t elements) then /* T is full */
6:              evict the bottom element of T using T.delete();
7:          create new element for id and put it at the top of T using T.insert(id);
```

The flow identifier $id = \mathrm{id}(m)$ of an incoming packet m will be placed in the cache T with probability $\alpha > 0$, that is, random_put(α) realizes a coin flip and returns true with probability α. If the cache T does not contain the identifier id, then the operation T.insert(id) inserts id at the top of the cache T; otherwise it moves the id to the top of T. When T contains t elements (i.e., T is full) and a new item is going to be inserted, the item at the bottom of T, the t-th element, will be evicted by the operation T.delete( ).

The randomized cache T determines which items are deleted from the queue in the case of congestion. If a flow with flow identifier id contains x packets, then the probability that its id will end up in the cache at some point in time is $1 - (1 - \alpha)^x$. Thus, processes sending more requests are more likely to be *inserted* into the cache. We will show that if the cache t is small compared to the length of the queue Q of pending packets, $t \ll q$, then a flow with many packets, an elephant, is much more likely to reside in the cache T than a mouse, a flow with few packets. If this queuing scheme is used in a router, then fewer users get penalized in the case of congestion, namely the ones that use up most of the bandwidth.

Remark. An obvious variation of our scheme is to store each flow identifier in T together with a counter. Initially, one inserts $(id, 0)$ into T, and if the flow identifier id is again inserted, then the flow identifier is moved to the top of the cache and the counter is increased by 1. This way, one deletes first the packets of flows in T that have high counters until enough packets are purged from the queue to alleviate the congestion. In practice, however, this scheme does not differ significantly from our scheme because of Theorem 3 below and the fact that the cache is small.

3 Analysis

The list T realizes a randomized cache with least-recently-used update strategy. The purpose of this cache is to discriminate between elephant and mice flows. We assume that the flows submitting packets are given by a finite set U containing n elements. Furthermore, we assume that the requests arrive one by one and each new request has a probability of $\Pr[u]$ to be from flow u, independently of the previous requests. In other words, the requests of the flows $u \in U$ are independent and identically distributed with probability $\Pr[u] > 0$, $\sum_{u \in U} \Pr[u] = 1$. This assumption is a matter of convenience, it turns out that our scheme shows the same behavior for many other probability distributions, but we focus on the uniformly distributed case that is easier to analyze in view of the space constraints.

The cache is fairly small in an actual system, typically much smaller than the length of the pending queue, $t \ll q$. Therefore, the long term behavior of this cache is of particular interest. We assume that the cache T is of size $t \leq n$. Flow identifiers are inserted into the cache T with probability $\alpha > 0$. Since we are only interested in the long term behavior, we may assume that the cache contains t identifiers. Hence, a state of the cache can be described by a string of t letters over the alphabet U, which contains no repetitions. The set of all possible states of the cache T is denoted by S. We use a Markov chain M_α to model the behavior of the cache T. The states of M_α are given by the set S of all states of the cache. We have $\binom{n}{t}$ selections of flow identifiers in the cache, and $t!$ possible orderings, which gives a total of $\binom{n}{t} t! = n!/(n-t)!$ different states of the Markov chain.

The admissible transitions of the Markov chain reflect the move-to-front rule of the cache. Several components contribute to the transition probabilities of the Markov chain M_α: the insertion probability α, and the probability $\Pr[u]$ that flow u issues a request. A state $s = (u_1, \ldots, u_t)$ of the cache remains unchanged if a message is not included into the cache or if a request of u_1 is selected for inclusion into T; the transition probability $P(s, s)$ is therefore given by $P(s, s) = 1 - \alpha + \alpha \Pr[u_1]$. If a message by $u \in U$ is selected to be included into the cache T and u is not contained in T, then the Markov model makes a transition from the current state $s = (u_1, \ldots, u_{t-1}, u_t)$ to the state $s' = (u, u_1, \ldots, u_{t-1})$ with probability $P(s, s') = \alpha \Pr[u]$. If a message by flow u is selected, and u is already in T, but not at the top of the cache, then the Markov model makes a transition from the current state $s = (u_1, \ldots, u_\ell, u, u_{\ell+2}, \ldots, u_t)$ to the state $s' = (u, u_1, \ldots, u_\ell, u_{\ell+2}, \ldots, u_t)$ with probability $P(s, s') = \alpha \Pr[u]$.

In the Markov chain M_α there is a nonzero probability to go from one state to any other state (in t steps), hence M_α is irreducible. Since there is a nonzero probability to stay in the same state, M_α is aperiodic. It follows that there exists a limiting probability measure

π on S, which satisfies

$$(\pi(s) : s \in S)P = (\pi(s) : s \in S), \tag{1}$$

where $P = (P(s, s'))_{s,s' \in S}$ is the transition matrix of the Markov chain M. No matter in which state the Markov chain is initially, the sequence of states will approach this probability distribution [3].

Theorem 1 *The stationary distribution π on the state space S of the Markov chain M_α, with insertion probability $\alpha > 0$, is given by*

$$\pi(s) = \Pr[u_1] \prod_{k=2}^{t} \frac{\Pr[u_k]}{\left(1 - \sum_{\ell=1}^{t-k+1} \Pr[u_\ell]\right)}, \tag{2}$$

where s is the state $s = (u_1, \ldots, u_t)$.

Proof. We verify by direct calculation that the probability measure π given in (2) satisfies the stationarity condition (1). According to (1) and the transition rules of the Markov chain M_α, we find that $\pi(s) = \pi(u_1, \ldots, u_t)$ satisfies the equation

$$\begin{aligned}\pi(u_1, \ldots, u_t) &= (1 - \alpha)\pi(u_1, \ldots, u_t) \\ &\quad + \alpha \Pr[u_1] \left(\sum_{u \neq u_1, \ldots, u_t} \pi(u_2, \ldots, u_t, u) + \sum_{m=1}^{t} \pi(u_2, \ldots, u_m, u_1, u_{m+1}, \ldots, u_t) \right).\end{aligned}$$

The first term on the right hand side models the fact that the state remains unchanged if an arriving item m of flow u_1 is not included in T. The last two terms model all possible states of T, which lead to s after inclusion of m. Subtracting $(1 - \alpha)\pi(s)$ from both sides and dividing by α yields

$$\pi(u_1, \ldots, u_t) = \Pr[u_1] \left(\sum_{u \neq u_1, \ldots, u_t} \pi(u_2, \ldots, u_t, u) + \sum_{m=1}^{t} \pi(u_2, \ldots, u_m, u_1, u_{m+1}, \ldots, u_t) \right). \tag{3}$$

Clearly, it suffices to check that (2) satisfies (3). It will be convenient to denote by $d_i(s)$ the term

$$d_i(s) = 1 - \sum_{\ell=1}^{t-i+1} \Pr[u_\ell].$$

Substituting (2) for $\pi(u_2, \ldots, u_t, u)$ yields

$$\sum_{u \neq u_1, \ldots, u_t} \pi(u_2, \ldots, u_t, u) = \sum_{u \neq u_1, \ldots, u_t} \Pr[u] \prod_{k=2}^{t} \frac{\Pr[u_k]}{d_{k-1}(s) + \Pr[u_1]} = \sum_{u \neq u_1, \ldots, u_t} \Pr[u] \prod_{k=1}^{t-1} \frac{\Pr[u_{k+1}]}{d_k(s) + \Pr[u_1]}.$$

Note that the sum $\sum_{u \neq u_1, \ldots, u_t} \Pr[u] = d_1(s)$. Since the product term on the right hand side does not depend on u, it follows that

$$\sum_{u \neq u_1, \ldots, u_t} \pi(u_2, \ldots, u_t, u) = d_1(s) \prod_{k=1}^{t-1} \frac{\Pr[u_{k+1}]}{d_k(s) + \Pr[u_1]}.$$

Similarly,

$$\sum_{m=1}^{t} \pi(u_2, \ldots, u_m, u_1, u_{m+1}, \ldots, u_t) = \sum_{m=1}^{t} \frac{\prod_{k=1}^{t} \Pr[u_k]}{\prod_{i=2}^{m} d_i(s) \prod_{i=m}^{t-1} (d_i(s) + \Pr[u_1])}.$$

Substituting the last two equations into equation (3) for $\pi(s)$, we find that

$$\pi(s) = \left(\frac{\prod_{k=1}^{t} \Pr[u_k]}{\prod_{i=2}^{t} d_i(s)} \right) \left[\frac{\prod_{i=1}^{t} d_i(s)}{\prod_{k=1}^{t-1} (d_k(s) + \Pr[u_1])} + \sum_{m=1}^{t} \frac{\Pr[u_1] \prod_{i=m+1}^{t} d_i(s) \prod_{i=1}^{m-1} (d_i(s) + \Pr[u_1])}{\prod_{i=1}^{t-1} (d_i(s) + \Pr[u_1])} \right].$$

We can simplify this expression to

$$\pi(s) = \left(\frac{\prod_{k=1}^{t} \Pr[u_k]}{\prod_{i=2}^{t} d_i(s)} \right) \left[\frac{\prod_{i=1}^{t} d_i(s) + \left(\Pr[u_1] \sum_{m=1}^{t} \prod_{i=m+1}^{t} d_i(s) \prod_{i=1}^{m-1} (d_i(s) + \Pr[u_1]) \right)}{\prod_{i=1}^{t} (d_i(s) + \Pr[u_1])} \right],$$

where we have used the fact that $d_t(s) + \Pr[u_1] = 1$. It turns out that the term in brackets is equal to 1; this is a consequence of the polynomial identity

$$\prod_{i=1}^{t} x_i = \prod_{i=1}^{t} (x_i + \lambda) - \lambda \sum_{m=1}^{t} \prod_{k=m+1}^{t} (x_k + \lambda) \prod_{\ell=1}^{m-1} x_\ell$$

that is easily proved by expanding the right-hand side and simplifying the expression. ■

Suppose that $\Pr[u_1] \geq \Pr[u_2] \geq \cdots \geq \Pr[u_n]$. The preceding theorem shows that the most likely state in the limiting probability measure is the state $(u_1, \ldots, u_t)$. Notice that all Markov chains M_α reach the same limiting probability measure π, regardless of the insertion probability α. The main difference is that the process will converge more slowly to this limiting distribution for small values of α.

Denote by $\Pr[u_i, m]$ the probability to find the identifier of flow u_i at position m in the cache in the stationary distribution. The following theorem gives a precise analytic result for this probability:

Theorem 2 *If the requests of the flows are independent and identically distributed, then the probability* $\Pr[u_i, m]$ *to find the identifier of flow* u_i *at position* m *in the cache in a stationary state is given by*

$$\Pr[u_i, m] = \Pr[u_i] \sum_{z=0}^{m-1} (-1)^{m-1-z} \binom{n-1-z}{m-1-z} \sum_{\substack{|Z|=z \\ u_i \notin Z}} [1 - Q_Z]^{-1}, \quad 1 \leq m \leq t, \tag{4}$$

for any inclusion probability $\alpha > 0$. *The inner sum is taken over all subsets* Z *of the set of flows* U, *and* $Q_Z = \sum_{u \in Z} \Pr[u]$.

Proof. Suppose that the cache is in a stationary state. Let the history of past requests, which have been selected for inclusion in the cache, be given by

$$(\cdots, u_{j_3}, u_{j_2}, u_{j_1}).$$

If the most recent request of flow u_i is request j_{k+1}, then u_i is at position m in the cache if and only if the set $X_k = \{u_{j_k}, \ldots, u_{j_1}\}$ has cardinality $m-1$. We use this simple observation to determine the probability for flow u_i to be at position m in the cache.

It follows from our assumptions that the requests u_{j_k}, which are included in the cache, are independent, and identically distributed. The request u_{j_k} occurs with probability $\Pr[u_{j_k}]$. According to our previous observation, flow u_i is at position m in the cache if and only if for some $k \geq 0$, $u_{j_{k+1}} = u_i$, and the random subset X_k of the past k requests does not contain u_i, and $|X_k| = m-1$. This allows us to express the probability $\Pr[u_i, m]$ in the form

$$\Pr[u_i, m] = \sum_{k=0}^{\infty} \Pr[u_{j_{k+1}} = u_i, u_i \notin X_k, |X_k| = m-1],$$

because the events in the brackets are disjoint for different values of k. We can state the right hand side more explicitly in terms of subsets Y of cardinality $m-1$ of the universe

U of flows:

$$\Pr[u_i, m] = \sum_{k=0}^{\infty} \Pr[u_i] \sum_{\substack{|Y|=m-1 \\ u_i \notin Y}} \Pr[X_k = Y] \quad = \Pr[u_i] \sum_{k=0}^{\infty} \sum_{\substack{|Y|=m-1 \\ u_i \notin Y}} \Pr[X_k = Y].$$

The inclusion-exclusion principle yields $\Pr[X_k = Y] = \sum_{Z \subseteq Y} (-1)^{|Y-Z|} \Pr[X_k \subseteq Z]$. In other words,

$$\Pr[X_k = Y] = \sum_{Z \subseteq Y} (-1)^{|Y-Z|} \Pr[u_{j_k} \in Z, \ldots, u_{j_2} \in Z, u_{j_1} \in Z] = \sum_{Z \subseteq Y} (-1)^{|Y-Z|} Q_Z^k,$$

where $Q_Z = \sum_{u \in Z} \Pr[u]$. Combining this expression for $\Pr[X_k = Y]$ with our previous formula for $\Pr[u_i, m]$ yields, after exchanging sums,

$$\Pr[u_i, m] = \Pr[u_i] \sum_{\substack{|Y|=m-1 \\ u_i \notin Y}} \sum_{Z \subseteq Y} (-1)^{|Y-Z|} \sum_{k=0}^{\infty} Q_Z^k.$$

A straightforward reformulation of this expression gives

$$\Pr[u_i, m] = \Pr[u_i] \sum_{\substack{|Z| \leq m-1 \\ u_i \notin Z}} \sum_{\substack{Z \subseteq Y \\ |Y|=m-1 \\ u_i \notin Y}} (-1)^{|Y-Z|} \left[1 - Q_Z\right]^{-1},$$

which can be simplified to

$$\begin{aligned} \Pr[u_i, m] &= \Pr[u_i] \sum_{z=0}^{m-1} \sum_{\substack{|Z|=z \\ u_i \notin Z}} (-1)^{m-1-z} \binom{n-1-z}{m-1-z} \left[1 - Q_Z\right]^{-1} \\ &= \Pr[u_i] \sum_{z=0}^{m-1} (-1)^{m-1-z} \binom{n-1-z}{m-1-z} \sum_{\substack{|Z|=z \\ u_i \notin Z}} \left[1 - Q_Z\right]^{-1}, \end{aligned}$$

which concludes the proof. ∎

Equation (4) shows, for instance, that flow u_i is found at the top of the cache with probability $\Pr[u_i, 1] = \Pr[u_i]$. Hence the flow with the most packets has the highest chance to be at the top of the cache.

For our application, we are particularly interested in the probability that an elephant flow can be found ahead of a mouse flow in the cache.

Theorem 3 *In the stationary distribution, the probability to find flow u_i ahead of flow*

u_j in the cache is given by

$$\Pr[u_i \text{ is ahead of } u_j] = \frac{\Pr[u_i]}{\Pr[u_i] + \Pr[u_j]},$$

regardless of the cache inclusion probability $\alpha > 0$.

Proof. Let us assume that all flows are initially in some total order. The move-to-front rule is applied when a flow is included in the cache, introducing a new order. This way, the first t flows represent the state of our randomized LRU cache, and the new order ensures that a flow which is evicted from the cache will be ahead of all flows outside the cache.

The probability to find flow u_i ahead of flow u_j after k requests is

$$\begin{aligned}&\Pr[u_i \text{ is ahead of } u_j \text{ after } k \text{ requests}]\\ &\quad= \frac{1}{2}(1 - \alpha \Pr[u_i] - \alpha \Pr[u_j])^k + \sum_{m=1}^{k} (1 - \alpha \Pr[u_i] - \alpha \Pr[u_j])^{k-m} \alpha \Pr[u_i],\end{aligned}$$

where the first term on the right hand side describes the case that u_i was initially ahead of u_j and neither were included in the cache during these k requests; the second term represents the case that u_i is included in the cache at time $m \geq 1$, and u_j was not included after time m. A straightforward proof by induction shows that

$$\begin{aligned}&\Pr[u_i \text{ is ahead of } u_j \text{ after } k \text{ requests}]\\ &\quad= \frac{\Pr[u_i]}{\Pr[u_i] + \Pr[u_j]} - (1 - \alpha \Pr[u_i] - \alpha \Pr[u_j])^k \frac{\Pr[u_j] - \Pr[u_i]}{2(\Pr[u_i] + \Pr[u_j])}.\end{aligned}$$

In the limit $k \to \infty$, we get

$$\Pr[u_i \text{ is ahead of } u_j] = \frac{\Pr[u_i]}{\Pr[u_i] + \Pr[u_j]}.$$

This proves the claim, since we assumed that u_i and u_j are in the cache. ∎

If an elephant flow u_e sends $c > 1$ times more requests than a mouse flow u_m, then the probability $\Pr[u_e \text{ is ahead of } u_m] = c/(1 + c)$, whereas $\Pr[u_m \text{ is ahead of } u_e] = 1/(1 + c)$. This shows that elephant flows are more likely to be near the top of the cache, and mouse flows are more likely to be evicted from the cache or to be outside the cache.

Remark. There exists an extensive literature on the move-to-front rule for sorting [4–8], which corresponds to the case $\alpha = 1$ and $n = t$. More general deterministic LRU caches, with $\alpha = 1$, have been studied in [9, 10]. Although the goal of these papers is usually to

estimate the expected computational cost or the expected cache miss ratio, one can learn valuable lessons from these classical works.

4 Conclusions

We analyzed our randomized caching algorithm using a Markov chain model and derived a closed form for its stationary distribution. We computed, in closed form, the probability that a flow resides at a certain position in the cache. Furthermore, we calculated the probability that a flow is ahead of another flow in the cache. Our results show that elephant flows are more likely to reside in the cache. In contrast to [2], we do not have to compare each incoming packet with all flow identifiers in the cache to achieve this result.

Acknowledgments. We thank Narasimha Reddy for introducing us to the concept of his partial state router [2]. The research of H.L. was supported by the University of Denver PROF grant 88197. The research of A.K. was supported in part by NSF grant CCF-0218582, NSF CAREER award CCF-0347310, and a TEES Select Young Faculty Award.

References

[1] C. Estan, G. Varghese, New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice, ACM Trans. Comput. Syst. 21 (3) (2003) 270–313.

[2] Smitha, I. Kim, A. Reddy, Identifying long-term high-bandwidth flows at a router, in: HiPC '01: Proceedings of the 8th International Conference on High Performance Computing, Springer-Verlag, London, UK, 2001, pp. 361–371.

[3] G. Lawler, L. Coyle, Lectures on Contemporary Probability, Student Mathematical Library, IAS/Park City Mathematical Subseries, AMS, 1999.

[4] W. Hendricks, The stationary distribution of an interesting Markov chain, J. Appl. Prob. 9 (1) (1972) 231–233.

[5] J. Bitner, Heuristics that dynamically organize data structures, SIAM J. Comput. 8 (1) (1979) 82–110.

[6] P. Burville, J. Kingman, On a model for storage and search, J. Appl. Prob. 10 (3) (1973) 697–701.

[7] R. Rivest, On self-organizing sequential search heuristics, Comm. ACM 19 (2) (1976) 63–67.

[8] J. McCabe, On serial files with relocatable records, Operations Research 13 (1965) 609–618.

[9] W. King, III, Analysis of demand paging algorithms, in: Information Processing 71 (IFIP Congress, Ljubljana, Yugoslavia, 1971), North-Holland, 1972, pp. 485–490.

[10] E. Coffman, Jr., P. Denning, Operating Systems Theory, Prentice Hall, Englewood Cliffs, NJ, 1973.