Approximate Models for General Cache Networks

Elisha J. Rosensweig
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003–9264
Email: elisha@cs.umass.edu

Jim Kurose
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003–9264
Email: kurose@cs.umass.edu

Don Towsley
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003–9264
Email: towsley@cs.umass.edu

***Abstract*—Many systems employ caches to improve performance. While isolated caches have been studied in-depth, multi-cache systems are not well understood, especially in networks with arbitrary topologies. In order to gain insight into and manage these systems, a low-complexity algorithm for approximating their behavior is required. We propose a new algorithm, termed *a-NET*, that approximates the behavior of multi-cache networks by leveraging existing approximation algorithms for isolated caches. We demonstrate the utility of *a-NET* using both per-cache and network-wide performance measures. We also perform factor analysis of the approximation error to identify system parameters that control the precision of *a-NET*.**

I. Introduction

Many systems employ caching as a means to reduce the load on access links and shorten access time to selected content. While the basic caching unit is a single cache, equipped with management policies, some systems make use of several caches linked together, allowing each cache to also forward requests to its neighbors when needed. Common examples of such systems are hierarchical web and file system caches. Recently [1][2], there have been proposals for multi-cache designs over networks of Internet-like scale and structure.

Determining the performance of a specific multi-cache system is extremely difficult. Even for the single, isolated cache using the popular LRU replacement policy, the complexity of exact models of cache contents and performance grow exponentially as a function of cache size and the number of files in the system [3], making them ineffective as exact modeling tools. For this reason, a more useful approach is to *approximate* the behavior of the caching system, allowing some measure of inaccuracy in return for simpler modeling techniques. This has been done for isolated caches and for some cache networks, namely cache hierarchies [4].

The main drawback of existing models for networked caches is their limited scope. They address hierarchical topologies (i.e., trees), in which the source of content is connected to a node at the top of the hierarchy (i.e., the root of the tree), and rely heavily on this structure when constructing the approximation. Furthermore, due to the complexity of these systems, the models are developed for small topologies (e.g., 2-level trees), and do not easily scale up for use in larger topologies. These limitations make existing solutions unapplicable when dealing with arbitrary topologies or large-scale cache networks. Instead, what is needed is an approach that can be applied to any topology.

In this paper, we present a novel **m**ulti-**c**ache **a**pproximation (MCA) algorithm, denoted *a-NET*, for a cache network with caches using the LRU replacement policy. The approach taken by *a-NET* is to decompose the problem and compute a **s**ingle-**c**ache **a**pproximation (SCA) for each individual cache in the network, where the request misses for each file at each cache are forwarded along the shortest path to the content source. The incoming request stream at each cache is thus reevaluated as a combination of both exogenous requests as well as portions of the miss stream of each cache's neighbors. *a-NET* is an iterative process, updating the incoming request stream at each cache and recomputing its miss stream (which then becomes part of the input stream to neighboring caches) using an SCA algorithm, until the entire network converges to a steady state. Unlike existing models, that develop topology-specific MCAs, *a-NET* can compute an MCA for any topology, regardless of structure or scale.

The contributions of this paper are:

- We develop *a-NET*, a novel MCA for general-topology cache networks, that utilizes the SCA algorithm described in [3].
- We demonstrate the behavior of *a-NET* for multiple topologies, and identify some key parameters that affect its performance. Specifically, we show that dependencies within the reference stream are the main cause of inaccuracy for *a-NET*, and that an increase in network connectivity can greatly reduce this problem.
- We construct a Markov model that expresses the inter-request distances in a cache miss stream for a range of arrival distributions. Considering that the miss stream of one cache becomes part of the incoming stream of one of its neighbors, we use this model to demonstrate the effects of non-IRM miss streams on the hit-ratio at neighboring caches.
- We evaluate the accuracy of *a-NET*, in terms of both per-cache and system-wide metrics. Individual caches are evaluated using the miss probability of each cache, while the performance of the entire system is measured in terms of the average number of hops a request traverses till content is located.

This work was supported in part by the National Science Foundation under CNS-0626874. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

The structure of this paper is as follows. In Section II we present *a-NET*, and motivate our topics of focus in this paper with an example of the performance of *a-NET*. In Section III we develop a general approach for factor analysis of the prediction errors of *a-NET*. We apply this approach to tree topologies, and show how this analysis can assist in determining how *a-NET* will perform in specific settings. Next (Section IV), we analyze one of the most likely causes for approximation error in *a-NET*, namely the *IRM-violation* in the cache miss stream, using a Markov chain model to explain certain properties of the prediction error in *a-NET*. Section V presents the performance of *a-NET* over a wide range of topologies, for both cache-specific performance metrics as well as network-wide metrics. Section VI presents a survey of related work on cache approximations and cache networks, and we conclude with a summary of our findings, and future work, in Section VII.

II. Approximating Cache Networks using *a-LRU*

A. Model Description and Problem Statement

We begin by describing the system of interest in this paper, namely cache networks. Let $G = (V, E)$ be a network of caches, $V = \{v_1, ...v_n\}$, $E \subseteq V \times V$. Additionally, let $F = \{f_1, ..., f_N\}$ be the set of files in the system. Each file is stored permanently at one or more public servers $S = \{s_1, ..., s_m\}$ that are attached to the network, each to one or more $v \in V$. For all $s \in S$ define $files(s) \subseteq \{1, ..., F\}$ to be the file indices stored at the server, s.t. $\left|\bigcup_{s \in S} files(s)\right| = N$. Also, for all $s \in S$ and $v \in V$ let $\chi(s, v) = 1$ iff source s is attached directly to v, and otherwise $\chi(s, v) = 0$. For simplicity of presentation, we assume for the rest of the paper that each file source is attached only to a single cache in the network, denoted v_s.

Let a *path* P be an ordered set of nodes $P = (v_{P_1}, ..., v_{P_j})$ such that $\forall 1 \leq i < j$ $(v_{P_i}, v_{P_{i+1}}) \in E$. Given a node v and a file f_i s.t. $i \in files(s)$, let $P_i^v = (v = v_{P_1}, ..., v_{P_j} = v_s)$ be the shortest path from v to the source of f_i, v_s, where distance is measured in the number of hops. In case of a tie, one path is selected at random when the system is initialized and is maintained hereafter.

At each node in this system, a Poisson stream of file access requests arrives exogenously. A request for f_i is denoted req_i. For all $1 \leq i \leq N$, $v \in V$, $\lambda_{i,v}$ is the rate of exogenous requests for file f_i at node v. When a request for file f_i arrives at a cache v, it generates a *hit* if the file is located at the cache and a *miss* if not. In the event of a miss, the request is forwarded to the next-hop cache along P_i^v, or to s if $\chi(s, v) = 1 \wedge i \in files(s)$. In the event of a hit, the file is forwarded along the reverse path taken by the request, and cached at each node along the way. If the cache is full, one of the files in the cache is evicted to make room for the new file. Following common practice (e.g., [3], [4], [5]), we assume that all files have the same size, and so the cache size $|v|$ can be expressed in terms of the number of files it can hold at any given moment.

For a given path P_i^v, let $P_i^v[j]$ be the j-th node in the path, s.t. $P_i^v[1] = v$ and $P_i^v[2]$ is the next hop from the originating node v. Given two nodes $v, v' \in V$, define

$$R(v, v') = \{i : v' = P_i^v[2]\}.$$

This is the set of all request ids i for which v' is on the next hop from v along the shortest path to source s s.t. $i \in files(s)$. Let $r_{i,v}$ be the combined incoming rate of requests for f_i at node v, and let $m_{i,v}$ be the miss rate of f_i at node v. Then the rate of requests at each node can be expressed as

$$r_{i,v} = \lambda_{i,v} + \sum_{v' : i \in R(v', v)} m_{i,v'} \tag{1}$$

Note that while $\lambda_{i,v}$ is a Poisson stream of exogenous requests, the miss stream of a cache is not, and so when we refer to $r_{i,v}$ as the incoming rate at the node we are referring to the *average* rate of requests.

The miss rates at each node depend on several factors, in addition to the cache management policies. One of these is the time it takes to retrieve content to the cache after a cache miss. In this paper, we follow common practice [3][4], and assume that the file download time after a cache miss is significantly smaller than the inter-request timescale. Thus, once a cache miss occurs, the file is assumed to be instantaneously downloaded into the cache.

Our goal in this paper is to develop an algorithm that can predict, with high accuracy, the incoming and miss streams at each of the caches, given the exogenous incoming rates. With such an algorithm available, cache network developers can evaluate the performance of the cache network itself efficiently. To determine the incoming and miss streams, we must solve the system of equations (1), for all i and v, and this in turn requires determining the miss rate for each file at each node. To this end we developed *a-NET*, our MCA algorithm, that tackles this problem.

B. From a-LRU to a-NET

In [3], Towsley and Dan developed an SCA algorithm for LRU and FIFO caches. As explained shortly, *a-NET* uses this LRU SCA algorithm, denoted *a-LRU*, to compute an MCA for the entire network. We begin, therefore, with a short description of *a-LRU*.

Let $\vec{p_v} = (p_{1,v}, \ldots, p_{N,v})$ be the steady-state incoming request distribution for files in F at a certain cache v. *a-LRU* can be thought of as a function $contents(\vec{p_v}, |v|) = \vec{q_v}$, where $\vec{q_v} = (q_{1,v}, \ldots, q_{N,v})$ is the vector consisting of the probability for each file to be present in the cache at a random point in time.

a-LRU was developed for request streams that conform to the **I**ndependent **R**eference **M**odel, or IRM for short, which means that the probability that the i-th request will be req_i, given past requests, is still $p_{i,v}$. For a request stream conforming to IRM, we prove in the technical report [6] that

$$m_{i,v} = r_{i,v} \cdot (1 - q_{i,v}). \tag{2}$$

Given our discussion thus far, we can now define out MCA algorithm, *a-NET*, an algorithm for solving the following set

of equations, for each $v \in V$:

$$r_{i,v} = \lambda_{i,v} + \sum_{v':i \in R(v',v)} m_{i,v'} \quad (3)$$

$$p_{i,v} = \frac{r_{i,v}}{\sum_{j=1}^{N} r_{j,v}} \quad (4)$$

$$\vec{q_v} = contents(\vec{p_v}, |v|) \quad (5)$$

$$m_{i,v} = r_{i,v} \cdot (1 - q_{i,v}) \quad (6)$$

Eq. (3) is identical to Eq. (1), which combines the exogenous request stream with the miss streams of the neighbors of v to get the incoming request stream at each cache. Eq. (4) defines $p_{i,v}$ as the relative portion of requests for f_i at v. Eq. (5) repeats the definition of the $contents(\cdot,\cdot)$ function, and Eq. (6) is identical to Eq. (2). Regarding this last equation, note that it was proven only for IRM streams. When the request streams consist also of the miss streams of neighbors, that do not conform to IRM, the equation may not accurately predict the miss rate. This issue is studied in detail in sections III & IV.

a-NET solves these equations iteratively. Initially, we set $m_{i,v} = 0$ for all i and v. Note that in the order these equations are presented, each equation relies only the values of the exogenous rates $\lambda_{i,v}$, which we assume are given, and information available from previous equations. In each iteration, *a-NET* solves equations (3) through (6) in order, for all caches, using the output of the previous iteration as input to the next. *a-NET* halts when a predefined precision threshold ϵ is met, that indicates that little has changed over the last iteration. In our implementation, we used the mean-square distance between the rates of all the nodes and files. While in theory it is possible that this procedure will not converge to a system-wide solution, such a scenario has never occurred during the course of our use of the algorithm.

C. *a-NET performance - an example*

To motivate the issues we discuss in the coming sections, we present here an example of the performance of *a-NET*. When using an isolated cache, a commonly used performance metric is the *miss probability* of the cache, with lower values indicating a more useful cache. Approximations for these caches are then evaluated by observing the **M**iss **P**robability **R**atio (MPR) between the predicted and actual behavior [3].

We simulated the behavior of a 10-by-10 torus cache network. The exogenous request distribution and rate is the same for all caches, and requests are distributed according to Zipf distribution with parameter 1 (i.e., the i-th most popular file has $p_i = \frac{1/i}{\sum_{j=1}^{N} 1/j}$), as has been observed in real web access traces [7]. There are $|F| = 500$ files in the system, cache sizes are $|v| = 50$, and there are 4 sources of content $s_1 - s_4$ with each file is stored at exactly one of them. We compared the behavior of the system to that of the approximation generated by *a-NET*, and the MPR per node is plotted in Figure 1.

Two main features stand out in Fig. 1. First, *a-NET* consistently *under-estimates* the number of misses that occur at each cache. Second, in the MCA produced by *a-NET* does not err

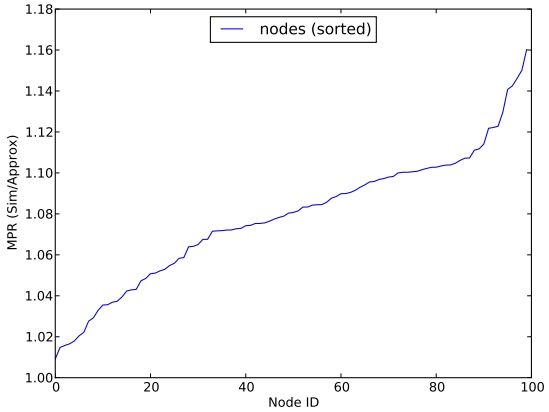

Fig. 1. MPR for 10-by-10 torus, with file requests distributed using Zipf distribution, $|F| = 500$, $|v| = 50$.

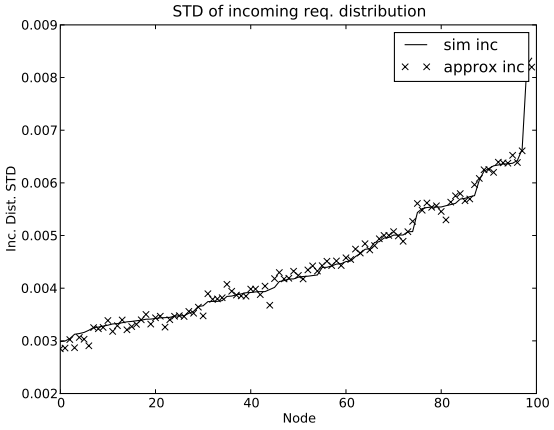

Fig. 2. Standard deviation of incoming request distribution at each node (sorted according to simulation values). The similarity of STD suggests that *a-NET* provides a good estimate of the distribution of requests at each node, despite under-estimating the MPR.

by more than 16% . These results indicate that, even though the IRM assumptions are invalid for cache networks, *a-NET* predicted performance do maintain some relationship to the simulated system performance. Support for this direction can be derived by plotting the standard deviation of the incoming file request distribution at each cache node, approximation vs. simulation, as in Fig. 2. The similarity of STD in each node indicates that the approximation is using as input at each node a steady-state distribution that is close to the actual distribution observed at that node. This would suggest that the cause for increased MPR lies elsewhere. Thus, in the sections to come we will more closely investigate what factors cause the errors we see here, and under what conditions are these errors minimized, turning *a-NET* into a useful MCA algorithm.

III. *a-NET* ERROR DECOMPOSITION

In this section we analyze the sources of errors in *a-NET*. Our goals here are (a) to determine the possible causes of

error; (b) to show how these errors can be distinguished; and (c) to use this information to determine in which scenarios *a-NET* will provide accurate predictions.

We investigate three potential causes for prediction error by *a-NET*:

1) **Inherent Prediction Error in the underlying SCA algorithm.** Since *a-LRU* only approximates the cache behavior, even with the correct request distribution the results may be skewed compared to the actual system behavior.
2) **Violation of IRM assumption.** The violation of IRM may adversely affect the performance of *a-NET* in two ways. First, *a-LRU* was designed only for IRM streams. Second, the miss rate of each file is calculated using Eq. (2), which is proven only for IRM streams. The usage of both of these outside of their natural context may cause *a-NET* to err.
3) **Input Error (or: propagating error).** The input given to *a-LRU* for cache v during *a-NET* execution includes outputs predicted by *a-LRU* for all of v's neighbors. Since the prediction of *a-LRU* is inaccurate, the request distribution given as input to *a-LRU* at each cache may not be the actual distribution at that cache. This can produce inaccurate predictions.

We would therefore like to determine the contributions of each of the errors in a given scenario, for which we have a simulation (denoted SIM) and the *a-NET* approximation (denoted APP). We disentangle the different errors from each other by extracting, from the simulation results, the actual steady-state distribution of the request stream at each node, and then evaluating the per-cache performance in two additional scenarios:

1) **Per-cache simulation with IRM traffic,** denoted SIM-IRM. Here, we simulate the behavior at each cache using the file request distribution extracted from SIM, but with file requests being generated from this distribution according to IRM.
2) **Per-cache approximation with simulation-driven traffic,** denoted APP-DRIVEN. Here we evaluate the cache performance using *a-LRU* at each cache, using the file request distribution extracted from SIM. Recall that the *a-LRU* algorithm assumes IRM arrivals.

We shall refer to these as the *pseudo-simulation* and *pseudo-approximation* respectively. In [3] the authors demonstrate that *a-LRU* gives close to optimal results for common scenarios, and so our goal here is to determine what part of the approximation error is due to IRM violation, and what part is due to input error, as defined above. We do so by comparing the predictions of *a-NET* in these different scenarios. Specifically,

- Comparing SIM to APP provides the MPR performance of *a-NET*, as in Fig. 1.
- Comparing SIM to APP-DRIVEN isolates the influence of IRM violation on the performance of *a-NET*, since the input at each cache is the same for both simulation and pseudo-approximation.
- Comparing SIM-IRM to APP-DRIVEN removes both the effects of IRM violation and input error, since the input at each cache is the same and the arrival streams at caches in SIM-IRM conform to IRM.

Factoring the error of *a-NET* in a specific scenario into its components can help develop improved prediction algorithms, that address these errors, as well as determine which cases will be more prone to prediction errors. We demonstrate this second point next for the case of cache trees.

A. Cache Trees

When using shortest path routing, cache trees form in every cache network that has a single source of content. Each node forwards its entire miss stream along a single link, the one on the shortest path to the source. Leaves are therefore nodes whose input stream does not include the miss stream of any neighboring cache. For the purpose of this case study, we consider only complete k-ary trees. As in the example case from Section II, we assume that (a) all caches are of the same size, and (b) the exogenous request stream at each cache is the same. These assumptions are maintained throughout this paper.

We initially consider the case of a linked list of h caches $v_0, \ldots, v_{h-1}$, with cache v_{h-1} linked to the single source (Fig. 3). We simulated the behavior of the system with parameters $h = 10$, $|v| = 50$, $N = 500$, where the exogenous request distribution is zipf with parameter 1.0. We generated, additionally, the pseudo-simulation and pseudo-approximation as described above, and plotted the pair-wise MPR between each of the simulations (standard (SIM) and pseudo (SIM-IRM)) , and each of the approximations. The results of these comparisons are presented in Figure 4.

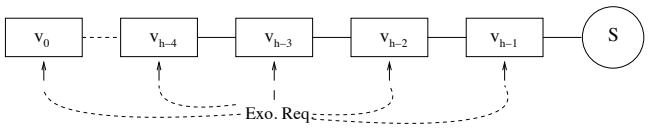

Fig. 3. k-ary tree for k=1, with request streams arriving exogenously at each node.

Several conclusions may be drawn from Figure 4. First, we note that when both the input error and IRM-violation errors are removed, the performance of the algorithms is close to optimal (a ratio of 1.0), which is strong indication that there are no additional hidden causes for error. Second, as in the example from Section II, the error leads *a-NET* to consistently under-estimate the probability of misses. Finally, it is clear that the input error is negligible in the case portrayed in Fig. 3, and that IRM-violation is the main source of error, which increases at caches closer to the source. This supports the observation made at the end of Section II, that *a-NET* provides a good estimate of the incoming request distribution at each node. Thus, one would expect that in scenarios where IRM is violated to a lesser degree, the performance of *a-NET* would improve.

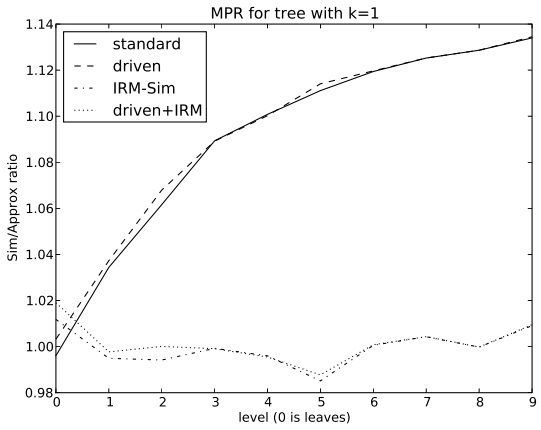

Fig. 4. Miss probability ratio for standard (SIM / APP), driven (SIM / APP-DRIVEN), IRM-Sim (SIM-IRM / APP) and driven + IRM-Sim (SIM-IRM / APP-DRIVEN). Performance is better when closer to 1.0. When IRM-violation is removed, the performance of *a-NET* at all caches is close to optimal

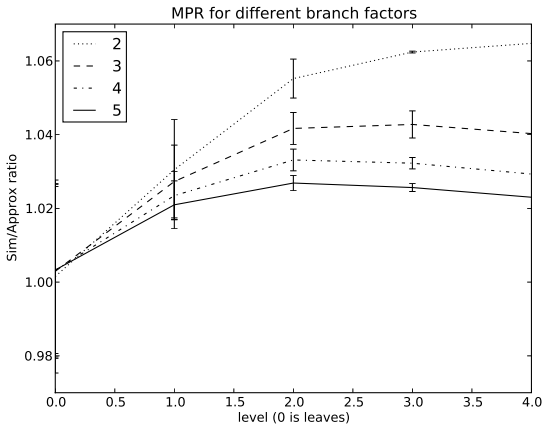

Fig. 5. Miss probability ratio for trees with branch factor $k = 2, 3, 4, 5$.

Support for this last hypothesis can be obtained from the performance of *a-NET* for larger trees, as we increase the branch factor k of the tree. As k grows, the incoming request stream at upper-level caches become more IRM-like. To understand why, consider a node A with two child nodes B and C, each of which is the root of its own sub-tree. Requests coming in to B are independent of those coming in to C since their respective sub-trees do not have any node in common. Therefore, the miss streams of B and C are independent of *each other*. However, the miss stream of B at each epoch depends on the state of cache B in that epoch. It is expected, therefore, that the aggregate of these two streams is in some sense closer to IRM than each of them would be on its own. As k goes to infinity, we get closer to a purely IRM request stream at A.

Based on this insight, we expect the performance of *a-NET* to improve as k grows. The results of testing this hypothesis for $k = 2, 3, 4, 5$ are shown in Figure 5. For each level in the tree, we calculated the MPR for each cache in that level, then plotted the mean MPR at that level with a 95% confidence interval using the student t-distribution. Our results show clearly that as the branch factor increases, so too does the accuracy of *a-NET*. We tested the error composition for additional distributions over tree topologies - uniform, truncated arithmetic and geometric - and consistently observed this behavior.

When we turn our attention to general topologies, similar behavior occurs as the average node degree grows. This is demonstrated and discussed in detail in Section V.

IV. Understanding IRM Violation in Cache Networks

As we've just seen, IRM-violation can cause *a-NET* prediction errors. In this section we present insight derived from an analytical model supporting the observation that *a-NET* generally underestimates the miss probability.

Let us begin with some intuition. It has been shown (for example, see [8]) that a chain of LRU caches performs poorly. One of the reasons for this is the lack of *locality of reference* in the miss streams of caches. When a cache miss occurs at v for file f, the file is downloaded into the cache, and so in order for another cache miss to occur it first must be evicted from the cache. Until this eviction occurs, the miss stream will not see another request for this file. This causes requests for file f to be, on average, farther apart in the miss stream than they are in the incoming stream, reducing the effectiveness of next-hop caches. This behavior can also help explain why *a-NET* consistently tends to underestimate the miss probability of caches. The inter-arrival distance between requests for f_i are likely to be greater in non-IRM miss-streams than in IRM streams. Since *a-NET* predicts misses assuming IRM, the miss probability it predicts will be lower than actually exhibited.

While the behavior just described is intuitive, and has been demonstrated empirically for many scenarios, to the best of our knowledge there is little analytical support for it. In this section we present a Markov model for the simple (and tractable) case of an IRM request stream with a distribution close to uniform, as defined shortly. We use this Markov model to demonstrate the effects of IRM-violation on the miss stream, and extrapolate from our results here to the effects of LRU caches on arbitrary distributions.

A. Description of Markov chain

We use a discrete Markov chain, to model the behavior of a tagged file f_* at cache v. We assume the arrival stream is IRM, that $p_{*,v} = \alpha$, and for all other files $p_{i,v} = \beta$, s.t.

$$\beta = \frac{1-\alpha}{N-1}$$

We are interested here in the non-IRM traffic characteristics of the miss stream. Following the general approach laid out in [8], we do so by measuring the *distribution* of the number

of requests between two req_*, termed here the *inter-miss distances*.

Our Markov chain consists of states (i, j), where

- $0 \leq i < \infty$. This variable represents the number of cache misses that have occurred for files other than f_*, since f_* entered the cache.
- $j \in \{1, \ldots, |v|, \text{E(vict)}, \text{M(iss)}, \text{A(bsorb)}\}$. This represents the state of f_*. For $i \leq |v|$, f_* is said to be in the i-th location of the cache. Otherwise, the file might be evicted (state E) or requested after eviction, generating a cache miss (state M). Finally, once a cache miss occurs, we go into the absorbing state (state A).

The transition matrix T can be derived from the transition diagrams, shown in Figures 6 - 7, which can easily be verified to correctly model the state of f_*. We assume that the system starts in state $(0, 1)$, which reflects the state of the system after a cache miss for f_*, with f_* at the top of the cache. Each arrival of a new request at the cache causes a transition in the Markov chain. For each state s, the probability that the system is in state s after k arrivals is expressed by $T^k[(0, 1), s]$. Therefore, the probability of an inter-miss distance being h for some $h \in \mathbb{N}$ is

$$P(distance = h) = \sum_{k=1}^{\infty} T^k[(0, 1), (h, M)]. \quad (7)$$

For all practical purposes, we need to set a cap for the distances h we are interested in computing, as otherwise the transition matrix is of infinite size. We denote that cap M_{max}.

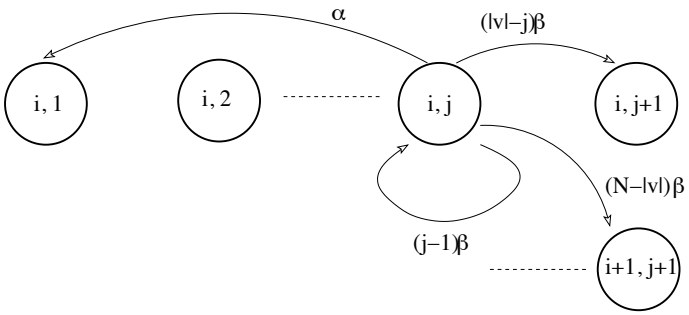

Fig. 6. Transition diagram for $1 \leq j \leq |v|$. For the case of $i = M_{max} + 1$, because state $(i+1, j+1)$ does not exist we give the transition to $(i, j+1)$ a total weight of $(N - j)\beta$.

B. Markov model results

Using the model described in Section IV-A, we analyzed several scenarios. For example, for the case of uniform distribution with $N = 30, |v| = 5, M_{max} = 80$, we compared the inter-request distance distribution of the tagged file, as computed by the Markov model, to the matching distribution for an IRM stream with the same request distribution for file request IDs. The cdf of these distributions is shown in Figure 8.

Let $G^M(x)$ be the inter-request distance distribution cdf for the non-IRM miss stream, as computed by the Markov

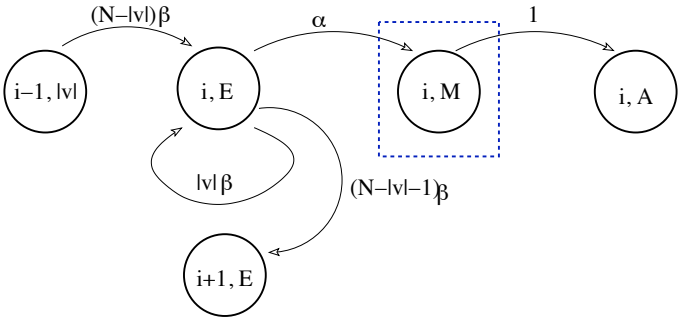

Fig. 7. Transition diagram for $j = E, M, A$. The probability of being in state (i, M) is the probability that the inter-miss distance is i. Because state $(i+1, E)$ does not exist, for the case of $i = M_{max}+1$ we give the transition to (i, E) a weight of $(N - 1)\beta$.

chain. Furthermore, let $G^I(x)$ be the cdf for an IRM request stream with the same distribution of files requested as the miss stream. As can be seen from Fig. 8, $G^M(x) < G^I(x)$ for all $x < N-1$. More generally, we make the following conjecture:

Conjecture 1: When the incoming request stream at a cache conforms to IRM, $G^M(x) \leq G^I(x)$ for all $x \leq |v|$.

This conjecture is motivated by the aforementioned fact that the tagged file f_* needs to be evicted from the cache before it can appear again in the miss stream, and so we expect the first $|v|$ misses following a miss for f_* to have a smaller probability of being requests for f_*, compared to an IRM model.

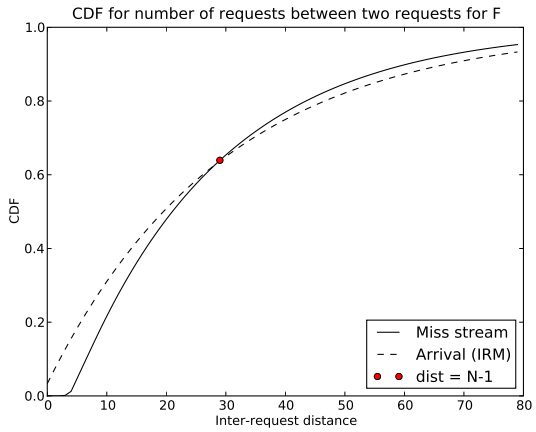

Fig. 8. CDF of inter-arrival distance, for both incoming (IRM) and miss (non-IRM) streams. The distance $N - 1$ marks the point after which the negative effects of the cache on the cumulative distribution are no longer felt.

We now proceed to use this Markov model to understand the effects IRM-violation in the miss stream will have on the performance of the next-hop cache. We present the following Theorem, proven in the technical report [6]:

Theorem 1: Let f_*, $G^M(k)$ and $G^I(k)$ be defined as above, and assume that each of the remaining files arrives with probability β. Additionally, let $Z^m(k)$ be the the inter-arrival distance distribution (IADD) cdf for all other files[1], and $Z^I(k)$

[1]We assume that these files have the same inter-arrival distribution. If not, $Z^m(k)$ takes the maximal value for each k over all the files.

be the IADD cdf for an IRM request stream with request probability β. Finally, let h_*^M and h_*^I be the hit probability of f_* at cache v for each model. Then, if $Z^m(k) \leq Z^I(k)$ for all $k \leq |v|$, then

$$h_*^M \longrightarrow_{N\to\infty} G^M(|v|-1)$$
$$h_*^I \longrightarrow_{N\to\infty} G^I(|v|-1).$$

Based on Theorem 1 and Conjecture 1, we can leverage the inter-arrival distance distribution generated by the Markov model to estimate the effects of non-IRM request streams on hit probability.

We present here our results from testing the case of $N = 50$, $|v| = 3$, for varying values of α, ranging from 0.05 to 0.8, and compare the resulting inter-miss distribution to the IRM equivalent in two ways. First, as mentioned above, we estimate the hit probability for each stream and plot the difference $G^I(|v|-1) - G^M(|v|-1)$. Second, we took the mean-square error of the two PDFs, to see how the hit probability was linked to overall distributional difference. The results of this experiment appear in Fig. 9.

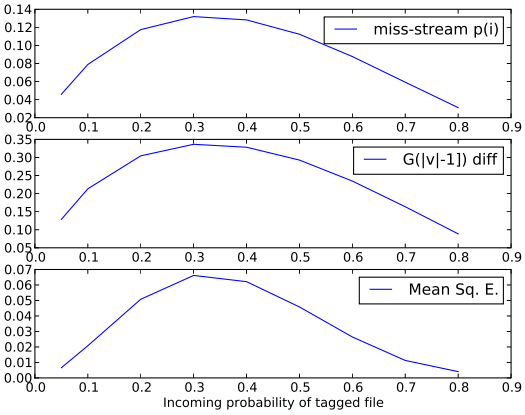

Fig. 9. Evaluating the effects of IRM violation on the inter-request distance distribution, as a function of the incoming probability of the tagged file ($p_* = \alpha$). *Top graph:* the fraction of requests for f_* in the miss stream. *Middle graph:* $G^I(|v|-1) - G^M(|v|-1)$. *Bottom graph:* the mean-square errors between the two PDFs, indicating how different the distributions are.

As can be seen here, there is a clear correlation between, the popularity of req_* in the miss stream, the hit probability difference and the mean-square error of the inter-arrival PDFs. As the popularity of the file in the *miss* stream grows (y axis in the top graph in Fig. 9), so does the difference between IRM and non-IRM distributions grow in terms of hit probability. Also, the changes in the distribution reflect this effect.

It is important to note that the error of *a-NET* in predicting the miss probability is proportionate to the popularity of the file. Specifically, in the case presented here, we get that $0.0224 \leq \frac{G^M(|v|-1)}{G^I(|v|-1)} \leq 0.0272$. Though popular files on the high end of this range, the ratios are very similar. This proportionality leads us to conjecture that the prediction error of *a-NET*, in terms of MPR, is mainly one of *scale*, but files retain their relative weight in the request streams at each cache. This conjecture is supported by the results in Fig. 2. We saw there that though the MPR can reach 1.16, the standard deviation in the incoming request streams is approximated with high accuracy by *a-NET*.

Another important conclusion to be derived from these results is that the performance of *a-NET* might differ for different files. Files on the two extremes of the popularity scale in the *arrival* stream will tend to suffer less from the imprecisions of *a-NET*, compared to files in the middle range. It is therefore reasonable to assume that metrics that break down performance based on content attributes, such as popularity, might be more suitable for analyzing MCA algorithms than cache-centric metrics, such as MPR. We present one such additional metric next, in the results section (Section V).

V. Numerical results

In this section we present numerical evaluations of the accuracy of *a-NET*. There are many parameters that affect the accuracy of *a-NET*, and so here we focus on several key parameters that greatly affect the performance of *a-NET*: network connectivity, cache size, request distribution and source clustering.

Connectivity. In accordance with our conclusions reached regarding cache trees, we tested the effects of increasing the node degree in a cache network on the MPR performance of *a-NET*. We generated random, 400-node graphs with each edge existing in the graph with probability $0.01 \leq p \leq 0.9$, with 500 files distributed according to Zipf distribution with parameter 1.0. Files were distributed between 10 sources randomly, and sources were randomly placed in each graph. The results are presented in Fig. 10. As expected, the MPR of *a-NET* gets closer to 1.0 as the connectivity of the graph increases. Note however that this improvement might be a result of the shorter distances between nodes, which limits the aggregation of errors in the prediction of *a-NET*. Further experimentation is required to determine the relative effect of these two factors. Whatever the exact cause is, however, increasing connectivity clearly improves the accuracy of *a-NET*.

Source Clustering. Another parameter that might affect performance is the clustering of the sources. When all sources are tightly clustered, the shortest path to each of the sources is the same for the most part from other nodes in the network. This creates a similar behavior to cache trees for the majority of nodes in the network; specifically, the network contains few *cross streams* - situations where a link or an entire path contains request streams flowing in opposite directions. In our work, we have observed that such occurrences tend to cause *a-NET* to have increased prediction error, and we are currently examining possible explanations for this phenomenon. We demonstrate it here over a 10-by-10 torus with 4 sources, and observed the mean MPR as the distance between sources grew. $|F| = 1000$, $|v| = 50$, files were assigned randomly to sources, and their arrival process was distributed using Zipf with parameter 1.0. The effects of clustering on the MPR

metric are shown in Fig. 11. As can be seen here, their is a slight but gradual increase in MPR as the average distance between sources grows.

Realistic topologies, request distributions, cache size. In order to evaluate the performance of *a-NET* over realistic topologies, we used the GT-ITM tool for topology generation, and generated transit-stub topologies modeled after the AT&T network, as suggested by Heckmann et al in [9]. Here, we compared the prediction of *a-NET* to the actual system using per-file average number of hops per request. This measure can be considered a system-wide metric, as it takes into account the interactions between caches. It is expected that *a-NET* will predict the number of hops with high accuracy for files on both extremes of the popularity scale, and less so for those in the middle. First of all, in Section IV we noted that the difference in hit probability is low on both of these extremes. Secondly, popular requests will mostly require a single hop, while unpopular requests will traverse the shortest path all the way to the source. Such performance can be clearly seen in Figure 12.

We considered the following scenarios, observing the effects of cache size and request distribution:

- $|F| = 500$, $|v| = 50$, distrib. = Zipf with parameter 1.0.
- $|F| = 500$, $|v| = 50$, distrib. = Zipf with parameter 0.6.
- $|F| = 500$, $|v| = 20$, distrib. = Zipf with parameter 1.0.
- $|F| = 250$, $|v| = 50$, distrib. = Zipf with parameter 1.0.

The results are presented in Figure 13. As can be seen here, *a-NET* performs better when the distribution is skewed to a larger degree, as when the zipf parameter is larger. However, in the examples shown here, the number of files in the system and the cache size seem to have little effect on performance.

For the topologies we experimented with here, the mean error is within range $10-15\%$. The diameter of the networks used here was ≤ 10, and so the errors seen here for hop count indicate predicting 1-2 hops less than actually occurs. Further experiments are required to determine how sensitive *a-NET* will be to an increase in network size.

We also observed the arrival distribution at all nodes for the simulations presented here, and calculated the mean square difference of the predicted vs. actual values. As expected, our results show that the mean-square error, averaged over all nodes, is $\leq 10^-6$, regardless of request distribution. This gives strong support to our conjecture that *a-NET* can provide reliable predictions for the incoming request distribution at each node.

VI. Related Work

In this paper we assumed exogenous request streams conformed to IRM. However, there are alternative models for request patterns at single caches. Panagakis et. al. [10] present approximate analysis for streams that have short term correlations for requests. In this model, the arrival process is IRM, with the exception that the k most recent requests have a higher probability of arriving next at the cache than other requests. The justification for this model is that such correlations have been found in web-traces. However, as we have seen, cache networks experiences the opposite effect, with recent requests *less* likely to arrive next, making this model inappropriate for such a system.

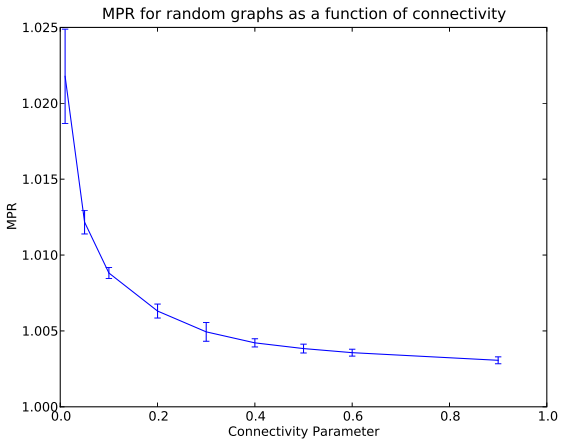

Fig. 10. Mean MPR for random graphs over 400 nodes, as a function of p, the probability that each edge is in the network. The mean is taken over 10 simulations for each p, with 95% confidence intervals showing.

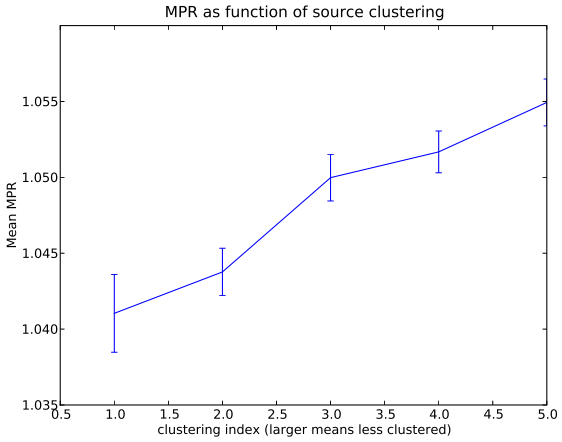

Fig. 11. Mean MPR for 10-by-10 torus networks, as a function of the source clustering. There were 4 sources positioned at the corners of a square, and the x-axis specifies the length of the square side in terms of no. of h ops. The mean is taken over 6 simulations for each clustering, with 95% confidence intervals showing.

Another alternative is Stack Depth Distribution (SDD) (for example, see [11]). With this model, the stream of requests is characterized as a distribution $\vec{h} = (h_i)_{i=1}^{\infty}$ over the cache slots in a cache of infinite capacity, where h_i is the probability that the next cache hit will be at slot i. In this model, all information regarding the individual files being requested is ignored or unavailable.

For IRM traffic, there are additional algorithms of equal complexity to *a-LRU* that compute the hit probability at a single cache, but these are not as easily used or as informative. For example, Flajolet et. al. [12] presented a integral solution for the cache approximation problem, which can be solved

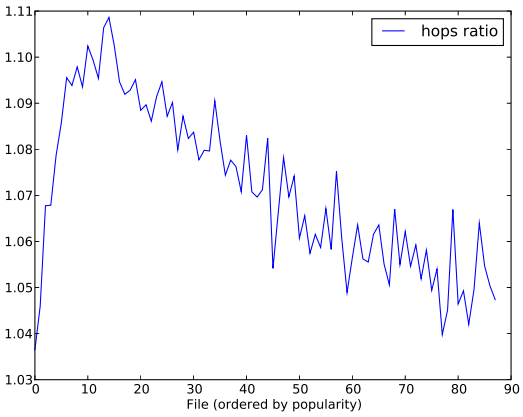

Fig. 12. Ratio of no. of hops for files stored at a given source. Files are ordered by popularity. As can be seen, highly-popular and highly unpopular files are approximated with greater accuracy.

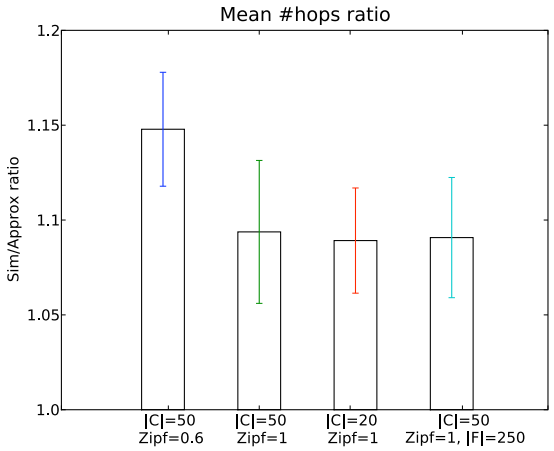

Fig. 13. Mean ratio of no. of hops per request. $|F| = 500$ unless specified otherwise. Results were obtained by random placement of 4 sources and random association of files with sources, over 6 simulation. Error bars represent 95% confidence intervals.

numerically to produce the hit ratio. However, there is no straight-forward manner by which to observe the behavior of each file with this approach.

Moving on to Multi-cache models, Che et. al. developed a model for a two-level LRU-based cache hierarchy [4], by using a "mean field" approximation of each cache. This approximation associates each file in each cache with a *constant* time, representing "the maximum inter-arrival time between two adjacent requests for [the] document without a cache miss". They justify this model by claiming that as the number of files in the system goes to infinity, this assumption becomes more reliable. Later on, this technique was leveraged in [5] to analyze cache coordination policies for cache hierarchies. Neither paper provide much simulation support for this model. Also, their approach is limited to 2-level cache hierarchies, and cannot be easily extended to larger tree sizes.

VII. Conclusions and future research

In this paper we presented *a-NET*, our MCA algorithm, which can be used to evaluate such performance measures as miss probability per cache and system-wide hops per request. In general, *a-NET* under-estimates the miss probability of caches, for which we have presented analytical support. The accuracy of our approximation is greatly affected by IRM-violations in the cache miss streams. It is thus most useful in situations where these affects are mitigated, as in cache trees with large branch factors and, in general, highly connected topologies. We have also seen that *a-NET* is highly accurate in predicting the incoming distribution at all nodes, even in cases where the MPR is large.

a-NET is designed to approximate the behavior of a cache network with static routing tables, while a more realistic model would consider dynamic routing as well. Adapting *a-NET* to such a model is a challenging task we plan to address next.

a-NET was presented here for LRU caches, but can be used with any SCA algorithm that receives the steady state distribution of request arrivals, and returns the probability of each file to be in the cache. The Markov model presented in section IV can also be adapted, with a small amount of changes, to other replacement policies, such as FIFO. We are currently working on several ways in which to leverage these properties model to expand our understanding of cache networks.

References

[1] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 161–172, 2001.

[2] E. Rosensweig and J. Kurose, "Breadcrumbs: efficient, best-effort content location in cache networks," in *IEEE INFOCOM Mini-Conference*, 2009.

[3] A. Dan and D. F. Towsley, "An approximate analysis of the lru and fifo buffer replacement schemes," in *SIGMETRICS*, 1990, pp. 143–152.

[4] H. Che, Z. Wang, and Y. Tung, "Analysis and design of hierarchical web caching systems," in *IEEE INFOCOM*, 2001, pp. 1416–1424.

[5] N. Laoutaris, H. Che, and I. Stavrakakis, "The lcd interconnection of lru caches and its analysis," *Performance Evaluation*, vol. 63, pp. 609–634, 2006.

[6] E. J. Rosensweig, J. Kurose, and D. Towsley, "Approximate models for general cache networks," UMass Amherst, MA, Tech. Rep. UM-CS-2009-037, 2009.

[7] L. Breslau, P. Cue, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *In INFOCOM*, 1999, pp. 126–134.

[8] R. Fonseca, V. Almeida, M. Crovella, and B. Abrahao, "On the intrinsic locality properties of web reference streams," in *In Proceedings of the IEEE INFOCOM*, 2003.

[9] O. Heckmann, M. Piringer, J. Schmitt, and R. Steinmetz, "On realistic network topologies for simulation," in *MoMeTools '03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, 2003, pp. 28–32.

[10] A. Panagakis, A. Vaios, and I. Stavrakakis, "Approximate analysis of lru in the case of short term correlations," *Comput. Netw.*, vol. 52, no. 6, pp. 1142–1152, 2008.

[11] H. Levy and R. J. T. Morris, "Exact analysis of bernoulli superposition of streams into a least recently used cache," *IEEE Trans. Softw. Eng.*, vol. 21, no. 8, pp. 682–688, 1995.

[12] P. Flajolet, D. Gardy, and L. Thimonier, "Birthday paradox, coupon collectors, caching algorithms and self-organizing search," *Discrete Appl. Math.*, vol. 39, no. 3, pp. 207–229, 1992.

VIII. Appendix

Lemma 1: With IRM traffic,

$$m_i = r_i \cdot (1 - q_i). \tag{8}$$

Proof: Let e_T be the identity of the T-th request in the incoming stream, and $m_{p,i}$ the miss probability of file i. By definition, for some random T,

$$\begin{aligned} m_{p,i} = & \quad P(e_T = req_i, f_i \notin C) \\ = & \quad P(e_T = req_i | f_i \notin C) \cdot P(f_i \notin C) \\ = & \quad P(e_T = req_i | f_i \notin C) \cdot (1 - q_i) \end{aligned}$$

and when IRM holds, the arrival probability does not depend on the state of the cache and so we get

$$P(e_T = req_i | f_i \notin C) = P(e_T = req_i) = p_i$$

Combining these results we conclude

$$\begin{aligned} m_i = r_{inc} \cdot m_{p,i} &= r_{inc} \cdot p_i \cdot (1 - q_i) \\ &= r_i \cdot (1 - q_i) \end{aligned}$$

■

A. Theorem 1 - statement and proof

Let $\vec{p_v}$ be the steady state distribution of requests arriving at cache v, where $p_1 = \alpha$ and for all other files $i \neq 1$ $p_i = \beta$. Let $g(k)$ be the pdf of the inter-arrival distribution of f_1, and $G(k)$ the cdf. Similarly, for each of the other files, let $z(k)$ be the pdf of the inter-arrival distribution, and $Z(k)$ the cdf[2] Finally, let $z^I(k)$ be the inter-arrival distance distribution for files f_i, $i > 1$, for the same $\vec{p_v}$ but when the stream conforms to IRM.

Lemma 2: Let $\eta_j^{(n)}(k)$ be the probability that the last k requests at the cache for stream n contained *less than* j requests for distinct files. Consider two uniform request streams, $S^{(1)}$ and $S^{(2)}$, such that in each stream, the inter-arrival distribution of each file is the same to all other files. Then, if for all $k = 0, ..., K$ we have

$$Z_1(k) \geq Z_2(k)$$

then for all $k = 0, ..., K$ we also have $\eta_k^{(1)}(k) \geq \eta_k^{(2)}(k)$.

Proof: Define (to simplify presentation) $Z_1(-1) = Z_2(-1) = 0$. Note that by definition, $\eta_k^{(i)}(k)$ is the probability of having *exactly* k distinct requests in k requests, which for a uniform file request distribution is

$$\begin{aligned} 1 - \eta_k^{(1)}(k) &= \prod_{i=0}^{k-1} (1 - Z_1(i-1)) \\ &\leq \prod_{i=0}^{k-1} (1 - Z_2(i-1)) = 1 - \eta_k^{(2)}(k) \end{aligned}$$

and so we get $\eta_k^{(1)}(k) \geq \eta_k^{(2)}(k)$ ■

[2]We assume that these files have the same inter-arrival distribution. If not, let $Z(k)$ take the maximal value for each k over all the files.

Proof: Note that $Z^I(k) \leq Z^I(k)$, so we prove the theorem here for h_*^M and h_*^I is a special case where the request stream is IRM. Let $\mu_i(k)$ be the probability that the previous k requests at the cache contained *less than* i requests for distinct files, given that a request for f_1 did not arrive in the previous k requests. Note that due to the condition the f_1 did not arrive, $\mu_i(k) \equiv \eta_i(k)$ where $\eta_i(k)$ is defined as in Lemma (2). Also, for all $k < |v|$, $\mu_{|v|}(k) = 1$, and that for all k $\mu_i(k) > \mu_i(k+1)$. By definition,

$$\begin{aligned} h &= \sum_{k=0}^{\infty} g(k) \cdot \mu_{|v|}(k) \\ &= \sum_{k=0}^{|v|-1} g(k) \cdot \mu_{|v|}(k) + \sum_{k=|v|}^{\infty} g(k) \cdot \mu_{|v|}(k) \\ &= G(|v|-1) + \sum_{k=|v|}^{\infty} g(k) \cdot \mu_{|v|}(k) \\ &\leq G(|v|-1) + \mu_{|v|}(|v|) \sum_{k=|v|}^{\infty} g(k) \\ &= G(|v|-1) + \mu_{|v|}(|v|) \cdot (1 - G(|v|-1)) \\ &\leq G(|v|-1) + \mu_{|v|}^I(|v|) \cdot (1 - G(|v|-1)) \end{aligned}$$

The last transition is based on the fact that $Z(k) \leq Z^I(k)$ for all $k \leq |v|$, and Lemma 2. We focus now on the second term. Note that the right hand side of it is ≤ 1. For an IRM request stream, we know

$$1 - \mu_k^I(k) = \prod_{j=1}^{|v|} \frac{N-j}{N-1}$$

and so we get

$$1 \geq 1 - \mu_{|v|}^I(|v|) = \prod_{j=1}^{|v|} \frac{N-j}{N-1} \longrightarrow_{N \to \infty} 1$$

which implies that

$$\mu_{|v|}^I(|v|) \longrightarrow_{N \to \infty} 0$$

and so we conclude that

$$h =_{N \to \infty} G(|v|-1) + 0 \cdot (1 - G(|v|)) = G(|v|-1).$$

■