A Network Calculus for 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

***Abstract*—Over the past few years Content-Centric Networking, a networking architecture in which host-to-content communication protocols are introduced, has been gaining much attention. A central component of such an architecture is a large-scale interconnected caching system. To date, the way these *Cache Networks* operate and perform is still poorly understood.**

Following the work of Cruz on queueing networks, in this paper we develop a *network calculus* for bounding flows in LRU cache networks of arbitrary topology. We analyze the tightness of these bounds as a function of several system parameters. Also, we derive from it several analytical results regarding these systems: the uniformizing impact of LRU on the request stream, and the significance of cache and routing diversity on performance.

I. Introduction

Today's Internet has become increasingly oriented towards content delivery. As a result, there has been growing interest in content-centric networks (CCNs) - networks in which content is addressable and host-to-content (rather than host-to-host) interaction is the norm [1]–[5]. In many CCN architectures, widespread caching plays a central, integrative role. While stand-alone caches have traditionally been used to bring popular content closer to content consumers (see [6]–[9] for a small sample of such discussions), the design, analysis and management of widely-deployed, tightly-connected, heterogenous Internet-scale *networks* of caches – referred to here *Cache Networks* and abbreviated "CNs" – is an important, yet relatively uncharted, field.

Analyzing the performance of cache networks is a daunting task. Individual caches have been relatively well-studied in isolation, and researchers have recently developed approximate performance models of cache hierarchies [8], [10], [11]. However, in the general case, content requests can flow in both directions along a link, and requests for different content may be routed differently, resulting in the merging and splitting of request streams and a consequently complex stochastic request-arrival process at each cache. The analysis of such general cache networks has only just begun [26].

In this paper, we develop a deterministic *network calculus* for computing bounds on the flows of content requests that arrive at, and depart from (in the case of a cache miss), caches in a general network of caches. We show how these flow bounds can then be used to calculate performance bounds for metrics such as the cache miss rate for a given piece of content at a given network cache. Our work is inspired by Cruz's pioneering network delay calculus [12] for deterministically bounding flows in general queueing networks, which later led to new bounding techniques [13]–[16] and found use in fields beyond classic queueing networks, including sensor networks [17], [29], smart grids [28] and anomaly detection [18]. While flows in a network delay calculus represent units of work routed among queues, flows in a cache network represent content requests rounted among caches. Here, a request may either be satisfied at a cache (and the content subsequently stored at downstream caches as requested content is returned to the requestor) or forwarded upstream to another cache in the event of a cache miss. Queueing networks and cache networks thus have many fundamental differences.

Our work makes several important contributions.

1) We define an upper-bound characterization of a stream of requests at a cache, and highlight differences between cache networks and queueing networks.
2) We develop a calculus for computing bounds on the miss stream of an LRU cache, given bounds on the incoming request stream. We show that these bounds are tight and consistent, i.e., that the upper bound can be realized for all files simultaneously.
3) We use this calculus to gain analytical insights into the behavior of LRU caches in isolation, and in networks. We identify the uniformizing effect of LRU on the request stream, and the impact of cache and topology diversity on system performance.
4) Using an iterative fixed-point procedure, we use this calculus to study LRU replacement in non-hierarchical cache networks. Our results indicate that these bounds can be close-to-tight for realistic network scenarios.

More generally, we believe our work represents an important step forward in developing performance models for emerging content-centric networks, as well as other systems in which an interconnected network of caches provides efficient, scalable content distribution.

The remainder of this paper is organized as follows. In Section II we discuss our network and flow models, and define the notion of bound *tightness*. In Section III we present theorems on bounding the number of cache misses over a *finite window*, and formulate theorems bounding the miss stream in Section IV. These theorems reveal analytical properties of LRU's impact on request flows. In Section V we use our calculus to study the performance of cache networks and evaluate bound tightness. We conclude with a review of related work and a summary of our results and discussion of future work.

II. A (ρ, σ) Model for Cache Networks

In this section we discuss the models we use here for cache networks (§II-A) and flow bounding (§II-B). We define the concept of *tight bounds* as used in this paper (§II-C), and conclude with an expositional example of bounds on the miss stream (§II-D).

A. Cache and Network Operation Model

We consider a network of nodes V inter-connected by edges, each node equipped with a cache for storing content passing through the node. Users connect to and send requests for content into the network. When the content is located, it is sent back to the requesting user, and stored in all the caches along the download path. We assume here that these pieces of content are of constant size (e.g., chunks of a file), and thus cache sizes are specified in terms of the number of pieces which the cache can store at any given moment. In this work, we shall refer to these content pieces as *files.*

We now discuss network operation in greater detail. Let $F = \{f_1, ..., f_N\}$ be the set of files users can request. A user that wants to retrieve a file $f_j \in F$ sends a request q_j into the network. The request then traverses the nodes of the network along a path that ends at a content *custodian* for f_j - a location which stores the content permanently, such as a content server. This path, termed the *search path*, is constructed according to some predefined routing policy. In what follows, we shall assume that every search path must eventually end up at a content custodian. As a result, content download is ensured within a finite amount of time.

When a request q_j arrives at v, the contents of v is inspected. If $f_j \notin v$, v logs the direction where the request arrived from, and if this is the only such entry for this content, forwards the request towards the content custodian. If this is not the only such entry, the request is not forwarded on, and when eventually the content passes through v, it is forwarded to all such logged directions and deletes the corresponding entries. In such a way content is forwarded *along the reverse search path* that each q_j used, until arriving at a requesting user. Along this path, it is *cached* in the caches along the way.

This architecture corresponds to that proposed for leading CCN architectures [3]. From an engineering standpoint, this architecture can be efficient, since other requests following the same request path will be satisfied sooner as the content is already being sent in their direction. From an analytical standpoint, this architecture ensures a one-to-one correspondence between the request and file flows at each v: for each request q_j leaving v there is a matching f_j that will arrive at v at some future point. Furthermore, when caches are full, for every forwarded cache *miss* there will eventually be a corresponding file *eviction* at v. The reason for this is that if q_j generates a miss, f_j will eventually be downloaded to v, which will evict some other file $f_h \in v$ to make room for f_j. Note that the next q_h arriving after this download takes place will result in a cache miss.

This last point highlights an important difference between queueing and cache networks. While with the former the number of packets leaving a node sums up to the number arriving (if we exclude dropped packets), in the latter the number of requests in the miss stream is strongly dependant on the traffic makeup of requests for other files. In this work we bound the degree to which this inter-file effect takes place.

Finally, we consider the issue of download delay. In real caching systems there is a time lag between when a cache miss occurs at v and when the requested file arrives at v, forcing an eviction. However, in the caching literature it is common to assume this download delay is negligible, and adopt a *zero-download delay* model, which we abbreviate here as ZDD [8], [19], [20]. This assumption is used in many works to achieve analytical tractability of the system. In this paper, we adopt this model as well; However, in Section III-C we argue that the bounds we compute for ZDD hold also when ZDD does not hold, making our results applicable to real systems.

Combining ZDD with downloading along the reverse search path, we know that files arrive at v in the same order in which their misses took place. As a result, with deterministic replacement policies (such as LRU), the order of requests arriving at the cache determines the miss stream at that cache. This observation is the basis for the analysis in the following sections.

B. Bounding Model

In this work we adopt the flow model proposed by Cruz [12]. For a stream of events over time let $R(t)$ be the number of events that took place at time slot t. These events can be jobs or packets in queueing networks, or content requests in cache networks. In this work, we consider the latter. For a stream $R(t)$, (ρ, σ) is a deterministic bounding representation of a stream, if for any interval $[t_1, t_2)$, $0 \leq t_1 \leq t_2 \in \mathbb{R}$,

$$\int_{t_1}^{t_2} R(t) \leq \lceil \rho(t_2 - t_1) + \sigma \rceil \tag{1}$$

Note that we take the ceiling of the bound since arrivals are binary in nature - a request either arrives or does not arrive during some window, yet $\rho(t_2 - t_1) + \sigma$ can be any real number.

In queueing networks, the standard interpretation of ρ is the average arrival rate per time unit, and σ indicates the "burstiness" component of the stream, as the bounds allow σ packets to arrive irrespective of the size of the window. In Cruz's work [12] this bounding property was denoted $R \sim (\sigma, \rho)$. However, as we shall see later on, cache networks differ from queueing networks in that the rate component dominates the impact on the miss stream. We express this by modifying the notation slightly, and use instead $R \sim (\rho, \sigma)$.

The key result in [12] for queueing networks is that if each input flow j (corresponding to a source-destination node pair in a queueing network) has a $(\rho_{j,in}, \sigma_{j,in})$ characterization, then its output flow has a $(\rho_{j,out}, \sigma_{j,out})$ characterization that can be computed as a function of the input characterizations at that node, $\{(\rho_{k,in}, \sigma_{k,in})\}_{k=1}^{N}$ for N input flows. These output flows are then the input flows at the subsequent network nodes, and in this manner, per-flow bounding characterizations can be "pushed" through feed-forward networks. For non-feedforward networks, a system of simultaneous equations can be established and solved. Here we extend this calculus to cache networks, specifically those employing LRU caches, which is the policy of choice for many CCN architectures.

C. Bound tightness

Next, we address how to select the bound for a given stream. Since (ρ, σ) is only an upper bound, there are an infinite number of bounds for any given stream: for example, if (ρ, σ) is a bound for $R(t)$, then for any positive $\Delta_\rho, \Delta_\sigma$ also $(\rho + \Delta_\rho, \sigma + \Delta_\sigma)$ is a bound for $R(t)$. Thus, we define the following concept of bound tightness:

Definition 1. For a given (ρ_j, σ_j) bound for f_j requests we will say that it is *globally-tight* if (a) $\frac{1}{t}\int_{t'=0}^{t} R_j(t') \to_{t\to\infty} \rho_j$, i.e., if ρ_j is the average rate of requests, and (b) if $\lceil \sigma_j \rceil \geq 0$ is minimized given that ρ_j.

Lemma 2. *ρ is minimized over all bounds when the bound is globally-tight.*

Proof: Let (ρ_g, σ_g) be the globally-tight bound, and (ρ, σ) any other bound for $R(t)$. By construction, $\frac{1}{t}\int_{t'=0}^{t} R(t') \to_{t\to\infty} \rho_g$. Additionally, using equation 1 we conclude our proof:

$$\begin{aligned}
\int_{t'=0}^{t} R(t') &\leq \lceil \rho t + \sigma \rceil \leq \rho t + \sigma + 1 \\
\lim_{t\to\infty} \frac{1}{t}\int_{t'=0}^{t} R(t') &\leq \lim_{t\to\infty} \rho + \frac{\sigma+1}{t} \\
\rho_G &\leq \rho
\end{aligned}$$

■

In what follows we shall prove that globally-tight bounds always exists for boundable flows, and these are the bounds we shall compute. We select these bounds because they support the interpretation of ρ as the long-term mean arrival rate, which is convenient in several contexts. For example, for a given arrival stream characterization, we can compute ρ by computing the mean arrival rate. Also, globally-tight bounds minimize the ρ component, yielding a tighter bound for large windows.

D. Bounds at work: an example

We conclude this section with a simple example of bounding a miss stream, to give the reader some intuition regarding the impact of caches on request streams, specifically w.r.t. how these streams are characterized using the (ρ, σ) model. We use this example to draw distinctions between the manner in which queueing networks and cache networks behave.

Consider a cache of size 1, and N arrival streams bounded as follows:

$$\begin{aligned}
R_{1,in} &\sim (\rho_{1,in}, 0), \;\; \rho_{1,in} > 0 \\
\forall 2 \leq j \leq N \;\; R_{j,in} &\sim (0, \sigma_{j,in}), \;\; \sigma_{j,in} > 0
\end{aligned}$$

Aside from the f_1 flow, the rest of these streams will consist of a finite number of requests over an infinite window of time. The miss stream is maximized for all request streams when the arrival stream is an alternating sequence of requests, i.e.,

$$f_1, f_{\neq 1}, f_1, f_{\neq 1}, f_1, f_{\neq 1}, \ldots$$

In this sequence, all q_j for $j \neq 1$ will generate a miss, and we will also have $\sum_{j=2}^{N} \sigma_{j,in} + 1$ misses for f_j (we add the 1 for the first request of f_1). After all these misses take place, all requests for f_1 will generate cache hits. Thus, we move from arrivals $R_{1,in} \sim (\rho_{1,in}, 0)$ to misses $R_{1,out} \sim (0, \sum_{j=2}^{N} \sigma_{j,in} + 1)$.

We glean several insights from this example. First, note that the ρ component has disappeared in the miss stream, and that a σ component (that did not exist in the input stream) has appeared instead. Thus, there is no conservation of flows in this model, nor is there no conservation of ρ or σ individually.

Another point is that the miss stream of f_1 is bounded by the combined arrival streams for the other files. A cache miss for f_1 occurs only if requests for other files caused f_1 to be evicted from v before the next f_1 request arrived. This underscores the difference between queueing and cache networks. In the former, an increase in traffic of flow i might decrease the rate of flow j by causing flow j packets to be dropped; In the latter, an increase in flow i can have the opposite effect, by causing evictions of f_j and subsequent additional misses.

One final and critical insight here is regarding the interpretation of σ. In the context of a queue, the worst case usually occurs when a large *burst* of jobs arrives at the queue at the same time. Thus, in queueing networks, the σ component is commonly referred to as the *burstiness* component. However, in the example we show here, the worst case is when the $\sigma_{j,in}$ requests for f_j arrive spaced out, to generate maximum misses at the cache w.r.t. f_1 and f_j. Additionally, note that the miss stream of f_1 has a positive σ component, despite the fact that the miss stream is only a thinning of the non-bursty arrival process. Thus, in cache networks a more convenient way to think of the σ component is as a set of requests, each of which can arrive *at any time* for any given window, without positioning constraints. Despite this change, in what follows we shall stick with the conventional terms and refer to ρ and σ as the rate and burstiness components, respectively.

III. Computing Worst-Case Bounds for finite windows

In this section, we describe how to bound the miss stream for each file over a window $w = [s, t)$. For each file we are given the number of requests that arrive during w, and then we compute a bound on the number of misses per file during w.

f_j/q_j	jth file / request for jth file
c, N	cache size and # of files
$\mathcal{T}_j$	f_j request stream
$\mathcal{T}$	combined arrival stream
$I(w,j)$	num. of arriving q_j's during window w for $\mathcal{T}$
$O(w,j)$	num. of q_j misses during window w for $\mathcal{T}$
$M_{w,j}$	max. num. of miss sets for f_j during w
M_w	max. num. of miss sets during w
$\hat{M}_j$	max. miss rate for f_j

Table I
TABLE OF NOTATION

A. Notation and Preliminaries

We begin with notation, as summarized in Table I:

- f_j is the jth file, and q_j is a request for f_j. The cache size is c and the number of unique files is $|F| = N$.
- $\mathcal{T}_j = (t_{j,1}, t_{j,3}, t_{j,3}...)$ is a (possibly infinite) monotonically increasing sequence of times for f_j requests.
- $\mathcal{T}$ denoted a sequence of file requests arriving at a specific cache. Formally, $\mathcal{T} = \{\mathcal{T}_j\}_{j=1}^N$.
- The *outcome* of a request for f_j at time t is a *hit* if at that time $f_j \in v$, and a *miss* otherwise.
- For a window w and request sequence $\mathcal{T}$, let $I(w,j)$ be the number of q_j's in w, and $O(w,j)$ the number of misses. Note that for deterministic cache replacement policies, the misses can be computed as a function of cache contents at the outset of w and $\mathcal{T}$.

Definition 3. For any file f_j, requests for f_i where $j \neq i$ are said to be *interfering* with respect to f_j.

Definition 4. A *miss set* $s \subseteq F$ for a cache of size c is a multi-set of requests for at least $c+1$ unique files, where we omit the dependence of s on c for notational convenience. A *miss sequence* $\overrightarrow{s}$ is an ordered miss set. A *miss sequence for* f_j $\overrightarrow{s_j}$ is a miss sequence containing one or more requests q_j, such that these requests make up the sequence suffix. (e.g., (q_1, q_2, q_3, q_3) is a miss sequence for f_3 when $c = 2$, but not (q_1, q_3, q_2, q_3)).

A miss sequence for f_j is so named for the following reason. For a window w, if the arrivals during w form a miss sequence, there is a single miss for f_j which occurs at the first q_j in w. Since a miss set is a multi-set, it has the following property:

Property 5. *After adding requests or removing duplicate requests from a miss-set X, it remains a miss-set.*

B. Bounds over window w

We begin with bounding $O(w,j)$, given $\mathcal{T}$. For a given window w, denote with W a partition of w, $W = \{w_1, ..., w_l\}$ s.t. $w = w_1|w_2|...|w_l$. ($|$ indicates concatenation).

Lemma 6. *For a given request sequence $\mathcal{T}$ and window w, $O(w,j)$ equals the maximal number of $w_k \in W$ for any partition W s.t. the requests in w_k form a miss sequence for f_j.*

Proof: With LRU, a cache miss occurs for f_j iff c interfering requests for pairwise-different files arrive at the cache between two consecutive requests for f_j. Including the first q_j request at the end of this sequence, we get a miss sequence, and any additional requests for f_j at the end of the sequence retain the definition as a miss sequence, yet generate only hits. ■

Note that any partition of w defines also a partition of the arrivals over w into disjoint sets. In what follows, we will say that miss-sets are disjoint if each is contained in a different window in some partition of w. We next consider the case where the exact sequence $\mathcal{T}$ is unknown, and we are only given the *number* of arrivals $I(w,j)$ over w for all $1 \leq j \leq N$. Then, for each arrangement of these arrivals, the miss sequence can be different. Denote

- $M_{w,j}$ as the maximum number of disjoint miss-sets for q_j in any arrangement and partition of arrivals over w.
- M_w as he maximum number of disjoint miss-sets in any arrangement and partition of arrivals over w.

Note that M_w is not necessarily equal to $\max_j M_{w,j}$. To see this, consider a case where $c = 2$ and a sequence of requests over w consisting of a single request for each of $f_1, ...f_9$. In this scenario, $M_w = 3$, while for all j $M_{w,j} = 1$.

Using these definitions, the following corollary of Lemma 6 follows immediately:

Corollary 7. *Given $I(w,j)$ for all $1 \leq j \leq N$, $O(w,j) \leq M_{w,j}$, and there exists a sequence $\mathcal{T}$ for which this bound is reached.*

Proof: From Lemma 6 we have an equality between the number of misses and the number of miss sequences for the given arrival sequence. Since $M_{w,j}$ is the maximal number of miss sequences for f_j in any arrival sequence, $M_{w,j}$ bounds the misses for f_j and is reachable for some sequence. ■

Next, in the main result of this section, we quantify $M_{w,j}$:

Theorem 8.

$$M_{w,j} = \min\{I(w,j), M_w\} \tag{2}$$

Proof: Assume there is an arrangement and partition for which there are M_w miss-sets. If $I(w,j) \leq M_w$, from the pigeonhole principle we can move q_j's so that each is in a different miss-set. If there is a miss-set with duplicates, from Property 5 it can be moved to a set with no q_j without changing the number of miss-sets. Thus we get $M_{w,j} = I(w,j)$. Otherwise $I(w,j) > M_w$, and using the same argument we can move q_j's between sets until each has at least one q_j, in which case $M_{w,j} = M_w$, which concludes our proof. ■

C. The (lack of) impact of download delay

In Section II-A we mentioned the ZDD assumption we rely on in this paper. In this section we prove that this assumption - that download after a cache miss is instantaneous - does not impact the generality of the bounds we compute here.

Theorem 9. *The upper-bound on the miss stream for a ZDD system is also a bound on the miss stream for non-ZDD systems.*

Proof: For a non-ZDD system, consider a sequence $\mathcal{T}$ arriving over a window w_1, and let $w = w_1|w_2$ be the time from when the requests arrived until all files requested in w_1 were downloaded. Note that between an outstanding request and corresponding download, requests for the same file have no impact on the miss stream (as they are not forwarded), so we ignore these intermediate requests temporarily. We therefore associate each forwarded miss with a corresponding subsequent file download.

Next, if we keep the file arrival sequence as-is but shift the misses forward in time to their corresponding file download time in w, this has no impact on the *number* of misses taking place during w. This is clear since with LRU cache state is only affected by file arrivals and cache hits, but not cache misses.

This new configuration generates the same number of misses for the requests originally arriving in w_1, while abiding by ZDD. From Lemma 10 (see below) we know that including the misses we ignored earlier cannot decrease the number of misses over w. Thus, the miss stream of this cache does not decrease as a result of abiding by ZDD. Furthermore, increasing the number of misses for this cache will cause an increase in arrivals at neighboring caches, which again by Lemma 10 does not result in lower miss bounds, which concludes our proof. ∎

IV. Computing (ρ, σ) bounds on the miss stream

In this section we leverage the bounding techniques for finite windows to generate (ρ, σ) bounds on the miss stream, given bounds on the incoming stream. From the following lemma, we know we can generate the worst-case miss process when the arrival process is tight with the bounds over w:

Lemma 10. *For all $1 \leq i, j \leq N$, $M_{w,i}$ monotonically increases with $I(w, j)$.*

Proof: Increasing $I(w, j)$ can only increase the number of disjoint miss sets that can be constructed. So, from Equation 2 we get that $M_{w,i} = \min\{I(w, i), M_w\}$ monotonically increases. ∎

We will therefore assume that $I(w, j)$ equals the bounds over w, and say that the bounds are *tight over w*. Let $\hat{M}_w$ be M_w when for all $1 \leq j \leq N$, $I(w, j)$ is tight over window w, and similarly regarding $\hat{M}_{w,j}$. Since we assume the bounds are tight, and from Equation 1 the only parameter that impacts the bounds is the window size, the following lemma directly follows:

Lemma 11. *If $|w| = |w'|$, $\hat{M}_w = \hat{M}_{w'}$ and $\hat{M}_{w,j} = \hat{M}_{w',j}$.*

A. Bounding $\rho_{j,out}$

To compute bounds on $\rho_{j,out}$, we first show that the $\{\sigma_{i,in}\}_{i=1}^{N}$ parameters do not impact $\rho_{j,out}$, as they constitute only a finite number of requests:

Theorem 12. *Let $\mathcal{T}, \mathcal{T}'$ be two request streams with corresponding sets of globally-tight bounds, $\{(\rho_{j,in}, \sigma_{j,in})\}_{j=1}^{N}$ and $\{(\rho'_{j,in}, \sigma'_{j,in})\}_{j=1}^{N}$, such that for all $1 \leq j \leq N$ $\rho_{j,in} = \rho'_{j,in}$. Let $\{(\rho_{j,out}, \sigma_{j,out})\}_{j=1}^{N}$ and $\{(\rho'_{j,out}, \sigma'_{j,out})\}_{j=1}^{N}$ be the corresponding globally-tight bounds on these miss streams. Then, for all $1 \leq j \leq N$, $\rho_{j,out} = \rho'_{j,out}$.*

Proof: W.l.o.g., assume $\sigma_{j,in} = 0$ for all $1 \leq j \leq N$, and consider the q_j miss stream over some window w. We use the subscripts $\mathcal{T}$ and $\mathcal{T}'$ to distinguish between the different streams. Since we assume that the bounds are tight over w, we set $I_{\mathcal{T}}(w, j) = \rho_{j,in}(t - s) + \sigma_{j,in}$ and $I_{\mathcal{T}'}(w, j) = \rho'_{j,in}(t - s) + \sigma'_{j,in}$. W.l.o.g. assume that these values are integers, so we drop the rounding operation. Then, the difference in the total volume of the arrival streams is

$$\begin{aligned}\Delta &= \sum_{j=1}^{N} I_{\mathcal{T}'}(w, j) - \sum_{j=1}^{N} I_{\mathcal{T}}(w, j) \\ &= \sum_{j=1}^{N} (\rho'_{j,in} - \rho_{j,in})(t - s) + \sigma'_{j,in} - \sigma_{j,in} = \sum_{j=1}^{N} \sigma'_{j,in}\end{aligned}$$

For a given sequence $\mathcal{T}$, each request can belong to at most a single miss-sequence w.r.t. f_j. Thus, if we maximize the number of miss-sequences for both streams, Δ bounds the difference between the number of misses - $\left|O_{\mathcal{T}'}(w, j) - O_{\mathcal{T}}(w, j)\right| \leq \Delta$. Since Δ is independent of the window size and $O_{\mathcal{T}}(w, j) = \int_{u=s}^{t} R_{j,out}(u)$ we get

$$\left| \int_{u=s}^{t} R'_{j,out}(u) - \int_{u=s}^{t} R_{j,out}(u) \right| \leq \Delta$$

$$\left| \frac{1}{t - s} \left(\int_{u=s}^{t} R'_{j,out}(u) - \int_{u=s}^{t} R_{j,out}(u) \right) \right| \leq \frac{\Delta}{t - s}$$

We take the limit for $t \to \infty$, and recall the definition of globally-tight bounds,

$$\left| \rho'_{j,out} - \rho_{j,out} \right| \leq 0$$

$$\rho'_{j,out} = \rho_{j,out}$$

∎

Based on Theorem 12, we shall assume in this section that the sigma components of all arrival streams are 0. We next turn to bound the rate of the miss stream - a (ρ, σ) version of Theorem 8.

Theorem 13 (ρ bounds)**.** *Let $\hat{M} = \lim_{|w| \to \infty} \hat{M}_w / |w|$. Then the globally-tight bound on the mean miss rate for f_j is*

$$\rho_{j,out} := \min\{\rho_{j,in}, \hat{M}\}$$

and there exists a $\sigma_{j,out}$ for which $(\rho_{j,out}, \sigma_{j,out})$ is a bound for the miss stream.

Proof: From Theorem 8 we know $M_{w,j} = \min\{I(w, j), M_w\}$. For tight bounds and $\sigma = 0$, $I(w, j) = \lceil |w| \rho_{j,in} \rceil$, which results in

$$\hat{M}_{w,j} = \min\{\lceil |w| \rho_{j,in} \rceil, \hat{M}_w\},$$

$$\min\{|w| \rho_{j,in}, \hat{M}_w\} \leq \hat{M}_{w,j} \leq \min\{|w| \rho_{j,in} + 1, \hat{M}_w\}.$$

Dividing by $|w|$ and taking the window size to infinity we get

$$\min\{\rho_{j,in}, \hat{M}\} \le \lim_{|w|\to\infty} \frac{\hat{M}_{w,j}}{|w|} \le \min\{\rho_{j,in} + \lim_{|w|\to\infty} \frac{1}{|w|}, \hat{M}\}$$

and using this sandwich argument we get

$$\lim_{|w|\to\infty} \frac{\hat{M}_{w,j}}{|w|} = \min\{\rho_{j,in}, \hat{M}\} \tag{3}$$

Finally, if we maximize the number of misses per window, we get that regarding the miss process

$$\begin{aligned} \int_w R_{j,out}(t) &= \hat{M}_{w,j} \\ \lim_{|w|\to\infty} \frac{1}{|w|} \int_w R_{j,out}(t) &= \lim_{|w|\to\infty} \hat{M}_{w,j}/|w| \end{aligned}$$

and by the definition of global tightness we get

$$\rho_{j,out} = \lim_{|w|\to\infty} \hat{M}_{w,j}/|w| \tag{4}$$

and from Eq. 3 and 4 we conclude that $\rho_{j,out} = \min\{\rho_{j,in}, \hat{M}\}$.

We have shown here how to compute the mean miss rate over an infinite horizon. However, our bounds must hold as well for *all* windows, and so we must demonstrate next that this bound can be used with a *finite* burstiness component for all windows. To show this, we prove next that $\hat{M}_{w,j}$ monotonically increase as $|w|$ grows, and since $\rho_{j,out}$ is computed for an infinite window the argument is proven. Let w, w' be two windows s.t. $|w| = k \cdot |w'|$ for some integer $k > 1$. Since we can assume the burstiness component is zero in the arrival streams, the number of requests arriving in w is exactly k times that of what arrives in w'. Denote $|w'| = \delta$ and $w = [s, t)$. Then,

$$\begin{aligned} \tfrac{1}{|w|}\hat{M}_{w,j} &\ge_{(*)} \frac{1}{|w|} \sum_{h=0}^{k-1} \hat{M}_{[s+h\delta, s+(h+1)\delta),j} \\ &=_{(**)} \frac{1}{k|w'|} \sum_{h=0}^{k-1} \hat{M}_{w',j} = \frac{1}{k|w'|} k\hat{M}_{w',j} = \frac{1}{|w'|}\hat{M}_{w',j} \end{aligned}$$

Inequality (*) is a result of the fact that we can construct miss sets for w by iteratively doing so for each $[s+h\delta, s+(h+1)\delta)$ separately. Equality (**) is based on Lemma 11. From this derivation we see that $\frac{1}{|w|}\hat{M}_{w,j}$ is monotonic with the increase of the window size. Thus we know that $\rho_{j,out}$ can be applied to every window for some constant σ, which concludes our proof. ■

Discussion. Theorem 13 reveals some interesting properties of LRU caches. As this theorem states, the bound $\hat{M}$ is the same for *all* files - in the worst case, LRU acts as a capping mechanism on the arrival flow, enforcing a *cutoff point* at $\hat{M}$. Arrival rates are only affected by the cache if they go above a certain value. The literature on LRU contains results indicating that in practice LRU conducts a sort of "low-pass filtering" [20] or "Majorization" [21]. Previous work on this subject showed this for actual behavior but limited to simulation-based conclusions, specific topologies or analytical models for a limited range of arrival distributions. Our results here prove this to be the case for worst-case *bounds* over *arbitrary* boundable flows and network topologies.

B. Computing $\hat{M}$ as a function of input bounds

In the previous section we demonstrated how the bounds per flow are a function of $\hat{M}$. In this section we present an algorithm for computing M_w and $\hat{M}$.

We begin with the case of a finite window w. With Algorithm 1 we can compute the value of M_w when the input is $x_j := I(w, j)$ for all $1 \le j \le N$.

Theorem 14. *Algorithm 1 returns M s.t. $\lfloor M \rfloor = M_w$.*

Proof sketch: The algorithm consists mainly of iterating over two steps, shown in lines 8 and 10 of Algorithm 1. Line 10 (and initially 5) bound the number of miss sets by dividing the number of requests by $c + 1$, the size of a miss-set. Since duplicate requests in such a set do not increase the number of misses, in Line 8 we remove such requests from our accounting. this ensures we get an upper bound. When the algorithm concludes, from the pigeonhole principle we show that each request can be a part of a miss-set, so the bound is tight. Next, we prove this claim formally.

Algorithm 1 Bounds($x_1, ..., x_N$, c).

```
 1: // For all the following, $1 \le k \le N$
 2: for $1 \le k \le N$ do
 3:     $y_k := x_k$
 4: end for
 5: $M := \frac{1}{c+1} \sum_k y_k$
 6: while $\max_k y_k > M$ do
 7:     for $1 \le k \le N$ do
 8:         $y_k := \min\{x_k, M\}$
 9:     end for
10:     $M := \frac{1}{c+1} \sum_k y_k$
11: end while
12: RETURN $M$
```

Lemma 15. *If with an arrival of $\{x_k\}_{1\le k\le N}$ we can construct M_w miss sets, then with $y_k := \min\{x_k, M_w\}$ requests for each k we can construct M_w miss sets as well.*

Proof: From Property 5 we know that for any miss set, after removing duplicate requests it remains a miss sequence w.r.t. the same file. Thus, considering only the cases where all miss-sets are strict sets and not multi-sets is sufficient. To generate M_w strict miss-sets, at most one request for each file can appear in each such set, so M_w bounds the number of requests for each file, which concludes our proof. ■

Lemma 16. *If $M \ge M_w$ and $y_k := \min\{x_k, M\}$, then $M_w \le \frac{1}{c+1} \sum_k y_k$.*

Proof: Since $M \ge M_w$, then from Lemma 15 we know that by using only y_k requests for f_k does not reduce the number of miss sets we can construct. Next, since the minimal size of a miss set is $c+1$ and the sets are disjoint, the lemma is proven. ■

Theorem 17. *Algorithm 1 returns M s.t. $\lfloor M \rfloor = M_w$.*

Proof: Denote with M the output of the algorithm. First we show that in each stage of the algorithm, $M \geq M_w$. At the initialization of the algorithm we have $y_k = I(w,k)$, and from Lemma 16 we know that in Line 5 $M \geq M_w$.

If we enter the "while" loop, in each iteration we reduce y_k in a manner which, according to Lemma 15, does not reduce the maximal number of miss-sets that can be constructed. We then update in line 10 the value M, which according to Lemma 16 bounds the number of miss-sets that can be constructed. Repeated application of these two steps will therefore not violate the condition $M \geq M_w$. Thus, regardless of entering the loop, we always get $M \geq M_w$, and since $M_w \in \mathbb{N}$, this implies $\lfloor M \rfloor \geq M_w$.

Next we show that $\lfloor M \rfloor \leq M_w$ by proving that when the algorithm halts $\lfloor M \rfloor$ miss sets can be constructed. By the loop exit condition (line 6) we know that $y_k \leq M$ for all $1 \leq k \leq N$ when the algorithm halts, and that (from line 10) $M = \frac{1}{c+1}\sum_k y_k$. From line 10 we know that $y_k = M$ for at most $c+1$ files, and since for all $1 \leq k \leq N$ $x_k \in \mathbb{N}$, taking the maximum in line 8 ensures at most $c+1$ files have non-integer y_k. Thus, rounding down all y_k will result in enough requests to construct $\lfloor M \rfloor$ miss sets. By the pigeonhole principle, since for all files $\lfloor y_k \rfloor \leq M$, we can construct M miss sets, which concludes our proof. ■

Next, we use Algorithm 1 to compute $\hat{M}$. This is done by applying the algorithm with ρ components as inputs, and no rounding operation.

Lemma 18.

$$\frac{1}{t}Bounds(\rho_{1,in}t, ..., \rho_{N,in}t, c) = Bounds(\rho_{1,in}, ..., \rho_{N,in}, c)$$

Proof: To show this, we note that in both lines $8, 10$ the t parameter has linear impact. In the first iteration:

$$\textbf{line 8: } y_k = \min\{\rho_{k,in}t, \frac{t}{c+1}\sum_k \rho_{k,in}\} = t\min\{\rho_{k,in}, \frac{1}{c+1}\sum_k \rho_{k,in}\}$$

$$\textbf{line 10: } M = \frac{1}{c+1}\sum_k \rho_{k,in}t = \frac{t}{c+1}\sum_k \rho_{k,in}$$

and in all subsequent iterations, this phenomenon repeats itself, and so we can extract the t variable from the input. ■

Theorem 19. $\hat{M} = Bounds(\rho_{1,in}, ..., \rho_{N,in}, c)$

Proof: Let t be the length of window w. We can get bounds on the miss rate with

$$\frac{1}{t}M_w = \frac{1}{t}Bounds(I(w,1), ..., I(w,N), c)$$

For the case where the arrival streams are tight over the window w we get

$$\frac{1}{t}\hat{M}_w = \frac{1}{t}Bounds(\lceil \rho_{1,in}t + \sigma_{1,in} \rceil, ..., \lceil \rho_{N,in}t + \sigma_{N,in} \rceil, c)$$

From Theorem 12 we assume the burstiness is zero, and using a sandwich argument as in Theorem 13 the rounding operation can be ignored as $t \to \infty$. Thus, from Lemma 18 we conclude

$$\hat{M} = \lim_{t\to\infty} \frac{1}{t}Bounds(\rho_{1,in}t, ..., \rho_{N,in}t, c) = Bounds(\rho_{1,in}, ..., \rho_{N,in}, c)$$

■

The following theorem is also a result of this algorithm:

Theorem 20. *Consider two adjacent caches A, B such that the arrival stream at B consists totally of the entire miss stream of A, and B is smaller or equal in size to A. Then the bounds on the miss stream in A are identical to the bounds on the miss stream in B.*

Proof: This can be determined from Algorithm 1. For equal sized caches, the value $\lfloor M \rfloor$ computed for cache A will be computed in Line 5, and the loop will not be entered, so the same cap will be used. This the miss stream is a result of this capping, the cache B miss stream is unaffected. For smaller caches, the cap will be higher, once again having no impact on the miss stream of B. ■

Theorem 20 emphasizes the importance of cache and flow diversity in the network: a next hop cache can benefit the system if it is of a larger size, uses different replacement policies or accepts miss flows from a multitude of neighboring caches.

C. Achieving bounds in parallel

Until this point, our discussion has focused on the upper bounds per individual file, rearranging the arrival order of the interfering requests to generate the worst case for some f_j. In this section, we show that in fact these bounds are tight also in combination - the worst case can be reached for *all* files simultaneously. We do so using a constructive proof: Algorithm 2 provides an arrangement of requests that generates the worst-case for all files.

In Algorithm 2, which considers a window w, we take as input the number of miss sets M and, for $1 \leq k \leq N$, $y_k = \min\{I(w,k), M\}$ as used in Algorithm 1. We show now that the arrangement the algorithm produces will generate misses for all y_k requests, for all k, in the case where the cache was empty at the beginning of w.

Theorem 21. *All y_k requests for f_k will be cache misses, for all $1 \leq k \leq N$.*

Proof: First note that each q_k is in a different miss sequence, by the pigeonhole principle and the fact that $y_k \leq M$. Next, denote with *index(i,k)* the position of q_k in s_i. If $q_k \in s_i \cap s_j$ and $i < j$, the algorithm ensures

Algorithm 2 GetMissSets($y_1, ..., y_N$, c, M).

```
1: $S = \emptyset$
2: for k=1 to M do
3:     $\vec{s_k} = \emptyset$ // Initialize empty sequence
4:     $S := S \cup s_k$
5: end for
6: $j = 0$
7: for k=1 to N do
8:     for $h = 1$ to $y_k$ do
9:         $s_j := s_j | q_k$ // "—" indicates concatenation
10:        $j := (j+1) \mod M$ // Next $q_k$ request will be in a different sequence
11:    end for
12: end for
13: RETURN $S$
```

$index(i,k) \in \{index(j,k), index(j,k)+1\}$. Thus, if we concatenate the sequences in the reverse index order, i.e.,

$$\vec{s_M}.\vec{s_{M-1}}.....\vec{s_2}.\vec{s_1},$$

then all requests for the same file will be spaced out by at least c interfering requests. The first requests are all misses since we assume an empty cache, which concludes our proof. ■

The algorithm just described arranges exactly y_k requests per file to generate the worst case, when in practice there are $I(w,k) \geq y_k$ arrivals during w. To address this, we can place each of the excess $I(w,k) - y_k$ requests for f_k adjacent to a q_k in the sequence produced by the algorithm. This will not change the number of cache misses for any file, as a miss sequence for f_j can have an arbitrarily-long suffix consisting of requests for f_j.

D. Bounding $\sigma_{j,out}$

We next consider the burstiness components of the miss streams, given the arrival stream bounds and $\rho_{j,out}$ computed in the previous sections. In a slight variation of what we did earlier, we define the following set:

Definition 22. An *eviction set* $e \subseteq F$ is a multi-set of at least c requests for unique files. A *eviction sequence* $\overrightarrow{e}$ is an ordered eviction set. We say this is an eviction set (sequence) for file j if $q_j \notin e$ ($q_j \notin \overrightarrow{e}$).

We further define similar concepts for eviction sets as we did for miss sets. $E_{w,j}$ denotes the number of eviction sets for j over w; $\hat{E}_{w,j}$ is $E_{w,j}$ when the arrivals are tight with the arrival bounds; and $\hat{E}_j = \lim_{|w|\to\infty} \hat{E}_{w,j}/|w|$. Since appending an eviction sequence w.r.t. j with a request q_j yields a miss sequence, it can be shown from Theorems 8 and 13 that

$$M_{w,j} = \min\{I(w,j), E_{w,j}\} \tag{5}$$

$$\rho_{j,out} = \min\{\rho_{j,in}, \hat{E}_j\} \tag{6}$$

In what follows we also use the following two sets for each j: $X_j = \{k \neq j : \rho_{k,in} < \hat{M}\}$, and $Y_j = \{1, ..., N\} \setminus (X \cup j)$.

Theorem 23 (σ bounds). **(a)** *If $\rho_{j,in} < \hat{E}_j$, then $\sigma_{j,out} = \sigma_{j,in}$.*
(b) *If $\rho_{j,in} > \hat{E}_j$, then $\sigma_{j,out} =$ Bounds$(\{\sigma_{k,in}\}_k \in X_j, c - |Y_j| - 1)$*
(c) *If $\rho_{j,in} = \hat{E}_j$, then $\sigma_{j,out} = \min\{\sigma_{j,in},$ Bounds$(\{\sigma_{k,in}\}_k \in X_j, c - |Y_j| - 1)\}$*

As with the rate component, we see here once again that the less-popular files are unaffected by the cache (as shown in part (a) of the theorem), contrary to the popular files.

Proof: We adopt an amortized analysis approach here: for purposes of computing the bounds, we attribute requests first to the rate component and the rest to the burstiness component of a given bound. We say that the first group are *rate related* while the second is *burstiness related*. In this regard, we note that for any file such that $\rho_{j,in} < \hat{M}$, the entire rate component is accounted for in computing $\hat{M}$. We can see this by observing that in Algorithm 1 increasing this $\rho_{j,in}$ slightly (e.g. to anything less than $\hat{M}$) will result in an increase of $\hat{M}$. On the other hand, if $\rho_{j,in} > \hat{M}$, parts of the rate component are not associated with any rate-related miss-set.

(a) Assume $\rho_{j,in} < \hat{E}_j$, then we know $\rho_{j,out} < \hat{E}_j$ from Eq. 6. Thus, there is a large enough window [s,t) over which $(\hat{E}_j - \rho_{j,out})(t-s) \geq \sigma_{j,in}$, where we can place each of $\sigma_{j,in}$ requests for q_j after a eviction-sequence w.r.t. j, resulting in additional $\sigma_{j,in}$ miss-sets for j. This yields a total number of misses of $\rho_{j,out}t + \sigma_{j,in} = \rho_{j,in}t + \sigma_{j,in}$, which is clearly bounded by the input, so it is tight.

(b) Assume $\rho_{j,in} > \hat{E}_j$, then from Eq. 6 we know $\rho_{j,out} < \rho_{j,in}$, so we have an infinite number of requests for f_j that are not in a rate-related miss-set. We now construct additional miss-sets for f_j by using the burstiness components. For each $k \in Y_j$ we have an infinite number of q_k arrivals also not in any miss-set, and all we require to complete a miss set is to add a request q_j and an additional $c - |Y_j|$ unique requests from X_j. The number of these is at most Bounds($\{\sigma_{k,in}\}_k \in X_j$, $c - |Y_j| - 1$).

(c) If $\rho_{j,in} = \hat{E}_j$, both bounds from the previous sections hold using the same arguments above. Since one bounds the potential of interfering requests and the other the requests for f_j, taking the minimum of both gives us the bound on the miss stream burstiness. ■

What is left is to compute $\hat{E}_j$. To this end, note that eviction sets for f_j are identical to miss sets, except that (a) they do not include q_j's and (b) they are of size c, not $c+1$. Thus, to compute $\hat{E}_j$ we once again use Algorithm 1, but with two changes:

- The input given is only for files $k \neq i$.
- In line 10 we substitute $1/(c+1)$ with $1/c$.

The arguments and proofs are identical to those shown for the proof regarding $\hat{M}$ and so are not detailed here.

V. Evaluation of Worst-Case Bounds

A. Extracting bounds from Trace Data

When testing the models we propose in this work, we will be evaluating them against simulator-generated traces. In this section we discuss how to compute the (ρ,σ) bounds for flows in the simulation, both exogenous (user-to-router) and endogenous (router-to-router).

For a given trace, and since we compute here *globally-tight* bounds, this can be computed in linear time with the length (in terms of the number of requests) of the simulation:

- ρ_j is the mean request rate for f_j.
- To compute σ_j, compute first $\sigma' = \max_{k\in\mathbb{N}} \frac{1}{t_{j,k+1}-t_{j,k}}$, where $t_{j,k}$ is the arrival time of the kth request for f_j. σ' is the highest observed arrival rate. We then compute σ by canceling out the mean rate component for that same time slot, so we get $\sigma = \sigma' - \rho(t_{j,k+1} - t_{j,k})$.

In addition to computing bounds based on trace data, we may want to compute the bounds on an arrival process based on it's stochastic properties. Computing the ρ component is once again identical to the mean arrival rate of these processes. Regarding the burstiness component, some processes do not have a deterministic bound (e.g., exponential distribution). In such cases, we can use a statistical bounding point: for some α, let σ_α be such that

$$Pr(t_{j,k} \leq t + \sigma_\alpha | t_{j,k-1} = t) = \alpha$$

for all $k \in \mathbb{N}$. Then, $\sigma_\alpha - \rho$ is the burstiness bound.

B. Bound tightness in practice

We next present several results concerning the performance of our calculus. As in the analytical sections, we focus on the ρ component, due to its centrality for system performance. As proven in this paper, the bounds hold for all the experiments we conducted.

For Figures 2-4, the topology we consider is a complete binary tree of depth 4, where level 0 is the root node, shown in Figure 1. By default, we assume 600 unique files can be requested exogenously. One of the benefits of our calculus is the ability to compute performance for non-hierarchical systems. Thus, we place two custodians at nodes v_7, v_{14}, and split the files between them. As a result, the path from v_7 to v_{14} experiences *cross-flows* - flows of requests going in both directions. We consider the number of cross-flows to be the minimum of rates in either direction across a link, so for hierarchical systems this number is zero.

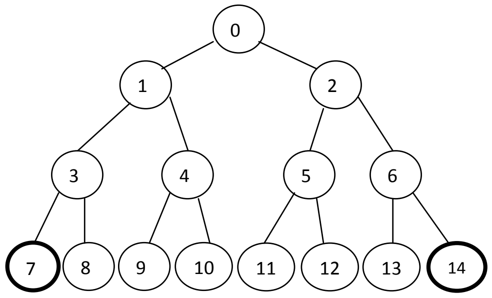

Figure 1. Topology for simulations. Custodians are at nodes $7, 14$.

Since we are interested in assessing the impact of cross-flows on the bound tightness, we consider the case where files are distributed according to multi-zipf distribution. The approach here is to divide the files into sets of equal size, give each set an equal probability, and then have the popularity within each set be distributed according to zipf. In the examples shown here we divide the files into eight sets of 75 files. The benefit of using this distribution is that it is uniform across the sets, so we can move sets between custodians and know that each set carries the same probability, while retaining the realistic scenario of non-uniform request patterns. Note that as the number of sets increases to N we get closer to uniform distribution, while as the number decreases to 1 we get the zipf distribution.

We consider two uses of our calculus. The first is for computing bounds on a network of arbitrary topology. We begin by computing bounds per node when the exogenous rates are the arrivals per node. These arrival rates are then recomputed by combining the exogenous rates with the bounds on the miss stream that are forwarded to that node. This process is then repeated until the system converges to a fixed point. We then compare the computed bounds to the actual performance of the system using simulations. As the bound-to-simulation ratio goes to 1, the bounds become more reflective of actual performance, indicating LRU performing close the it's worst case.

The second use of our calculus is for assessing the performance of LRU in cache network scenarios. In this context, we simulate the performance of a cache network and then extract the simulated arrival rates at each node. We feed these arrival rates to the calculus and compare the actual (simulated) miss rates with the bounds. The same interpretation of bound-to-simulation ratio applies here as well.

Figures 2-4 consider the first use case, where the computed bounds are fed to the next hop caches in the next iteration. In Fig. 2 we gradually shift content from the custodian at v_{14} to the one at v_7, which generates more cross-flows. We see in this figure how this increase in cross flows causes the bounds to be tighter, especially near the root of the tree (nodes 0-2) where the flows are largest.

In Fig. 3 we see how when decreasing the cache size the bounds become tighter, and the same phenomenon occurs when increasing the number of files. These results are especially relevant to Cache Networks, where the file-to-cache size ratio is expected to be high, making the bounding calculus a useful tool in estimating reasonable upper bounds on performance in practice.

We now turn briefly to applying our calculus in the second manner presented above, to determine how well LRU performs in a cache network. We once again consider the tree topology as before, but this time place a single custodian for all files at the root node, thus eliminating cross flows. The results are shown in Figure 5. They demonstrate that the bounds are getting tighter as we progress up the tree, indicating that cache hierarchies using LRU at all levels are inefficient as they increase in scale.

The importance of this calculus is highlighted when we consider a wider variety of arrival processes. Most models

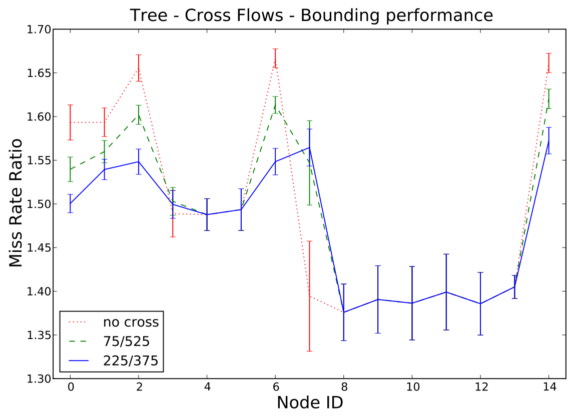

Figure 2. Impact of cross-flows on the bound tightness. cache size on bound tightness, with 90% confidence intervals shown. Setup is identical to Fig. 3. X/Y indicates X files at v_7 and Y files at v_{14}.

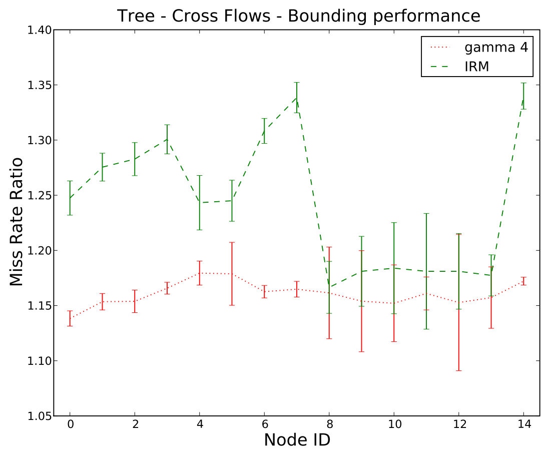

Figure 4. Impact of non-IRM traffic on bound tightness, with 90% confidence intervals shown. Setup is identical to Fig. 3. As we see here, with inter-arrival distances following the Gamma distribution with a scale parameter 4, bounds become more tight relative to with IRM.

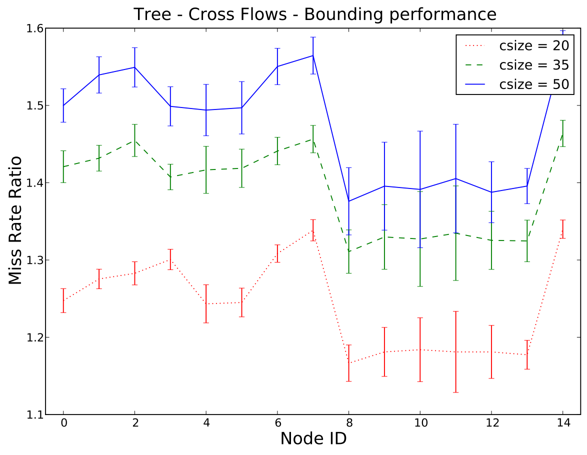

Figure 3. Impact of cache size on bound tightness, with 90% confidence intervals shown. Requests arrive at all nodes following a multi-zipf distribution. Files are divided between custodians at nodes 7, 14, with 225 files at the first and 375 at the second. As cache sizes decrease, bounds become more tight.

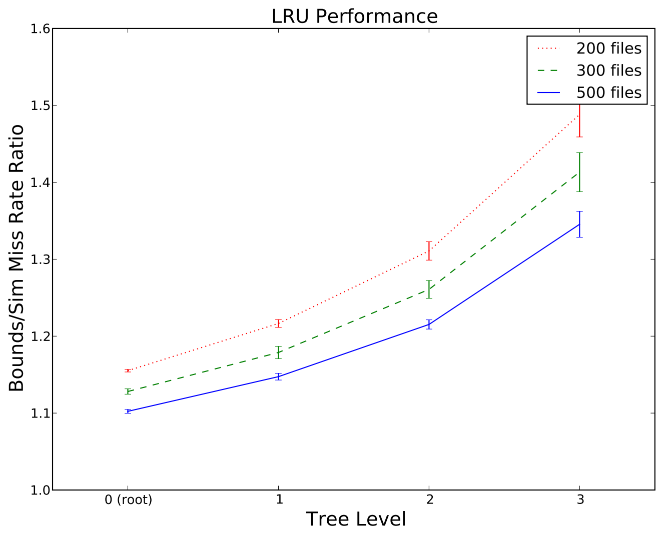

Figure 5. Performance of LRU as compared to LRU worst-case. 90% confidence intervals shown.

for caches consider only cases where the exogenous arrival process follows the Independent Reference Model (IRM). This means that requests are independent over time, such as when the inter-arrival time between every two requests for content f_j follows exponential distribution. In Figure 4 we show how varying the inter-arrival time distribution can generate worse performance for LRU. In this plot, we use the Gamma distribution to model exogenous arrivals: the time until the next arrival of a request for f_j with popularity $p_j = \lambda_j / \sum_i \lambda_i$ is modeled according to Gamma(p_j, β), where β is a scaling parameter ($\beta = 1$ generates the exponential distribution). As we can see in this figure, as the scaling parameter grows, the bounds become tighter. Thus, our bounding calculus is suitable for generating upper bounds in cases where the arrival process is not known in advance.

VI. Related Work

As we've seen in this paper, networks of caches exhibit extremely complex behavior. Even a single LRU cache in isolation with IRM assumptions are difficult to analyze exactly [19]. For individual LRU caches, the competitive ratio is known to be c, most of the modeling literature has analyzed performance for specific arrival processes to estimate the performance in practice [19], [22]–[24].

Given the complexity of single cache analysis, it is not surprising that cache network analyses to date have been *approximate* and limited to specific topologies. The modeling of networked caches has been considered mostly for hierarchies such as trees [8], [11], [20], [25], and recently for general topologies [26], which has shown the significant impact of dependencies within the miss streams on cache performance which existing models fail to capture.

An alternative approach to approximate analysis of complex networks is a bounding approach - the type of approach we have taken here. Beginning with Cruz's pioneering network delay calculus [12], numerous researchers have developed both deterministic and stochastic calculi for bounding the performance of networks of queues [13]. Networks of queues, where units of work proceed from one queue to another are quite different from networks of caches, which perform a filtering function, only forwarding cache misses on to downstream nodes.

A number of efforts have adopted a bounding approach, similar to network delay calculus for analyzing systems with complex, time-varying, stochastic flows. These efforts have analyzed energy flows in smart grid systems [27], [28] and energy harvesting/expenditure in wireless sensor networks [29]. While the flows in these systems are characterized by (σ, ρ) bounds, the behavior of individual components through which these flows pass, and the manner in which the bonded flows are transformed, are quite different from cache networks.

VII. Discussion and Future Work

In this work we presented a Network Calculus for boundable flows in an LRU cache network, and demonstrated its performance for non-hierarchical topologies which existing models do not address. Our bounds reveal that in the worst-case LRU acts as a cutoff point on the arrival process, giving analytical support to similar observations regarding actual behavior in the network.

The results presented here can be extended in several directions. The bounds here can be shown to hold equally well for FIFO, whose worst-case is very similar to that of LRU. Also, due to the limited impact of burstiness on the miss rate, similar bounds on the rate can be shown for non-deterministic bounding models, such as Exponentially Bounded Burstiness [15]. As for extending beyond deterministic replacement policies and addressing policies such as random replacement, a bound on the mean behavior would be more suitable, for which a different set of analytical tools will be needed.

References

[1] V. Jacobson. (2007) A new way to look at networking. Internet video. [Online]. Available: http://video.google.com/videoplay?docid=-6972678839686672840

[2] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A Survey of Information-Centric Networking (Draft)," in *Information-Centric Networking*, 2011.

[3] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '09. ACM, 2009, pp. 1–12.

[4] 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.

[5] D. Trossen, M. Sarela, and K. Sollins, "Arguments for an information-centric internetworking architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 40, April 2010.

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

[7] M. Busari and C. L. Williamson, "Simulation evaluation of a heterogeneous web proxy caching hierarchy," in *MASCOTS*. IEEE Computer Society, 2001, pp. 379–388.

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

[9] G. Peng, "Cdn: Content distribution network," 2004. [Online]. Available: http://arxiv.org/abs/cs/0411069

[10] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, and G. Pavlou, "Modeling and evaluation of ccn-caching trees," in *IFIP Networking*, 2011.

[11] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *INFOCOM, 2010 Proceedings IEEE*, march 2010, pp. 1 –9.

[12] R. Cruz, "A calculus for network delay. i. network elements in isolation," *Information Theory, IEEE Transactions on*, vol. 37, no. 1, pp. 114 –131, jan 1991.

[13] J. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. springer-Verlag, 2001.

[14] D. Starobinski, M. Karpovsky, and L. A. Zakrevski, "Application of network calculus to general topologies using turn-prohibition," *IEEE/ACM Trans. Netw.*, vol. 11, pp. 411–421, June 2003.

[15] D. Starobinski and M. Sidi, "Stochastically bounded burstiness for communication networks," *IEEE Transactions on Information Theory*, vol. 46, pp. 206–212, 1999.

[16] O. Yaron and M. Sidi, "Generalized processor sharing networks with exponentially bounded burstiness arrivals," in *Journal of High Speed Networks*, 1994, pp. 628–634.

[17] J. Schmitt and U. Roedig, "Sensor network calculus–a framework for worst case analysis," *Distributed Computing in Sensor Systems*, pp. 467–467, 2005.

[18] M. Raza, B. Robertson, W. Phillips, and J. Ilow, "Network calculus based modeling of anomaly detection," in *Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2010 International Symposium on*. IEEE, 2010, pp. 416–421.

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

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

[21] S. Vanichpun and A. Makowski, "Comparing strength of locality of reference-popularity, majorization, and some folk theorems," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 2. IEEE, 2004, pp. 838–849.

[22] W. F. King, "Analysis of paging algorithms," in *IFIP Congress*, 1971, pp. 485–490.

[23] 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.

[24] 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.

[25] Y. Zhou, Z. Chen, and K. Li, "Second-level buffer cache management," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 15, no. 6, pp. 505 – 519, june 2004.

[26] E. J. Rosnsweig, J. Kurose, and D. Towsley, "Approximate models for general cache networks," in *INFOCOM, 2010 Proceedings IEEE*, march 2010.

[27] K. Wang, F. Ciucu, C. Lin, and S. Low, "A stochastic power network calculus for integrating renewable energy sources into the power grid," *Selected Areas in Communications, IEEE Journal on*, vol. 30, no. 6, pp. 1037–1048, 2012.

[28] J.-Y. Le Boudec and D.-C. Tomozei, "Demand response using service curves," in *Innovative Smart Grid Technologies (ISGT Europe), 2011 2nd IEEE PES International Conference and Exhibition on*, dec. 2011.

[29] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power management in energy harvesting sensor networks," *ACM Trans. Embed. Comput. Syst.*, vol. 6, no. 4, Sep. 2007.