This paper was presented as part of the main technical program at IEEE INFOCOM 2011

Content-Aware Caching and Traffic Management in Content Distribution Networks

Meghana M Amble*, Parimal Parag*, Srinivas Shakkottai*, and Lei Ying†
*Dept. of ECE, Texas A&M University, Email: {amble38,parimal,sshakkot}@tamu.edu
†Dept. of ECE, Iowa State University, Email: leiying@iastate.edu

***Abstract*—The rapid increase of content delivery over the Internet has led to the proliferation of content distribution networks (CDNs). Management of CDNs requires algorithms for request routing, content placement, and eviction in such a way that user delays are small. We abstract the system of frontend source nodes and backend caches of the CDN in the likeness of the input and output nodes of a switch. In this model, queues of requests for different pieces of content build up at the source nodes, which route these requests to a cache that contains the requested content. For each request that is routed to a cache, a corresponding data file is transmitted back to the requesting source across links of finite capacity. Caches are of finite size, and the content of the caches can be refreshed periodically. Our objective is to design policies for request routing, content placement and content eviction with the goal of small user delays. Stable policies ensure the *finiteness* of the request queues, while good polices also lead to *short* queue lengths. We first design a throughput-optimal algorithm that solves the routing-placement-eviction problem. The design yields insight into the impact of different cache refresh policies on queue length, and we construct throughput optimal algorithms that engender short queue lengths. We illustrate the potential of our approach through simulations on different CDN topologies.**

I. Introduction

Recent years have seen the rise of the Internet as a means of content delivery [1], driven in part by the growing popularity of smart hand-held devices as a means of content consumption. Available content includes software and smart-phone applications, music and video files available for purchase, as well as media streaming applications. Each type of content is associated with a particular desired quality of service but, broadly speaking, low delays between request and reception is good for all types of content. A content distribution network (CDN) is a distributed system that routes requests for content arising from end-users to caches that can service these queries; the CDN then returns content using a network that connects such caches to end-users. The motivation behind such a system is that obtaining content from a cache that is near a user is likely to experience a shorter delay than from one that is farther away, due to a smaller number of hops to be traversed. However, placing a large demand for a popular piece of content on the nearest cache might be counterproductive, as the link capacity between each cache and the end-users is finite.

An abstraction of such a CDN is illustrated in Figure 1. On the *control plane*, it consists of frontend servers denoted by ‘S’, which aggregate queries arising in different geographical regions, and route each query to an appropriate backend cache indicated by a ‘D’. A frontend may have access to some subset of backend caches. Multiple backend caches can potentially serve each query, and each frontend has to take a decision on which such backend to pick. Further, cache sizes are finite and caches can be periodically refreshed from a media vault, along with eviction of stale content. On the *data plane*, the backend cache chosen to service a particular request needs to process the query, and transmit data across a network connecting it to the end-user. The following constraints affect system operation: (i) the network connecting the backend caches to the end-users has finite capacity, (ii) each backend cache can only host a finite amount of content, and (iii) refreshing content in the caches from the media vault incurs a cost.

Research was funded in part by NSF grants CNS-0904520, CNS-0963818, CNS-0963807, DTRA grant HDTRA1-09-1-0051 and the Google Faculty Research Awards program.

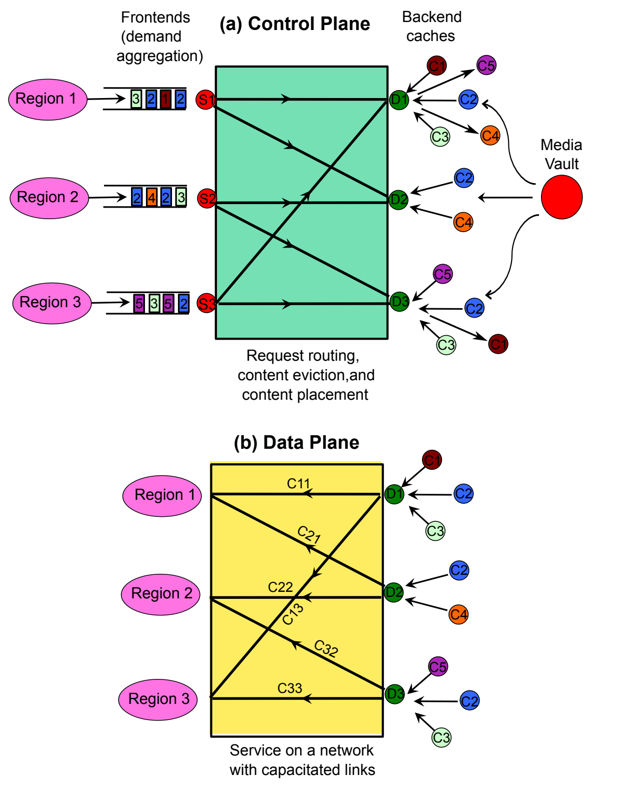


Fig. 1. A Content Distribution Network. (a) Control Plane: Requests arrive at frontend servers (S), and must be routed to one of (possibly) several backend caches (D) that have the content. Caches can only host a finite number of content files (C), and the caches may be refreshed by placement and eviction of content, (b) Data Plane: Content is served to end-users across a network consisting of finite capacity links.

In this paper we develop algorithms for jointly solving the request routing and content caching problems. The problem of content caching is related to an online paging problem [2], wherein requests are generated for different pages, either in a Bayesian fashion or by an adversary. A cache miss implies that the page has to be brought into the cache, which involves a cost of fetching. The objective is to decide what pages to evict when such misses happen, so as to minimize the total number of misses. However, there are a number of key differences in our content distribution scenario. According to our abstraction, requests are Bayesian with unknown statistics and queue up in a request buffer of infinite size. There is no possibility of a miss, but the queue lengths must be kept finite for system stability. A natural requirement of algorithms in such a scenario is throughput optimality, which means that any stabilizable request arrival vector should be stabilized by our algorithm. Further, a short queue implies a small service delay, and hence queue length is our quality metric. The cost of accessing the media vault is captured in the periodicity of refresh, with a larger periodicity implying a lower cost. Finally, we abstract the resource constraints of the network connecting caches to end-users by links of finite capacity—something that is missing in the paging problem. Our main goal is to design throughput optimal algorithms and provide quantitative performance analysis that could guide the design of distributed CDNs. We first review related work below.

Related Work

The problem of caching and eviction has been visited in the context of memory caches, on-line Web caching, and distributed Web storage systems. For example, [2]–[4] focus on online eviction algorithms (LRU, FIFO and LFU), and its variants such as greedy and randomized versions. They use misses and competitive coefficients as the cost metric. Load balancing and content placement with linear communication costs is examined in [5], [6]. The objective is to use distributed and centralized integer programming approaches to minimize such costs. However, they do not consider finite capacity links or the issue of online eviction decisions.

Our CDN abstraction bears a resemblance to the problem of scheduling in high speed switches. Tassiulas *et al.* proposed the Max Weight scheduling algorithm for switches and multi-hop wireless networks in their seminal work [7]. They proved that this policy is throughput-optimal, and characterized the capacity region as the convex hull of all feasible schedules. Various extensions of this work that followed since are [8]–[14]. These papers explore the delays in the system for single down-link with variable connectivity, multi-rate links and multi-hop wireless flows. While none of these directly applies to our problem, we will build upon the analytical techniques used in these papers as appropriate to our context.

Main Results

In this paper, we employ the abstraction of a switch architecture to develop algorithms for content distribution. We first develop a model of a CDN in Section II, under which a request queue implicitly determines the popularity of a piece of content. Content may be refreshed periodically at caches.

- We develop a *Periodic Max-Weight* (PMW) algorithm with random evictions in Section III, which follows from trying to stabilize a system using a Lyapunov function that is quadratic in the queue lengths. The algorithm consists of two parts: (i) fetch-evict-service at refresh instants, and (ii) service subject to presence of content at inter refresh instants. We show that the algorithm is throughput optimal, and has bounded queue lengths.
- We relate the average queue size to the drift of the Lyapunov function in Section IV, with larger negative drift implying a shorter queue length. The content that is evicted at refresh instants plays a major role in determining the drift at inter-refresh times. We use this insight to design a *Periodic Max-Weight Scheduling with Min-Weight Eviction* policy. We prove that it too is throughput optimal, and further it has low computational complexity.
- We observe that the above policies, although throughput optimal, are somewhat inefficient in their utilization of link capacity. In Section V we develop an *Iterative Periodic Max-Weight* algorithm that attempts to utilize all available link capacity, while ensuring that all service that would have happened with PMW still takes place. Again, this algorithm is throughput optimal, and is likely to have even lower queue lengths than PMW.
- Finally, In Section VI we illustrate the performance of all the algorithms through simulations on fully connected, and partially connected CDN topologies. We explore the changes in performance that arise through changes in refresh periodicity as well as cache size, and demonstrate the importance of appropriate algorithm selection in large refresh periodicity and small cache size regimes.

We conclude with ideas on future work in Section VII.

II. System Overview

The CDN consists of the set $\mathcal{S}$ of frontend nodes indexed by $s \in \mathcal{S}$ that serve as request sources, with $|\mathcal{S}| = S$. The set $\mathcal{D}$ of backend caches is indexed by $d \in \mathcal{D}$, with $|\mathcal{D}| = E$. The set of content files is denoted $\mathcal{C}$, with each $c \in \mathcal{C}$. We assume that request packets are small, and that request routing has no overhead. Hence, capacity constraints only apply to data delivery. Thus, supposing that content c is present at backend d, for any request being routed from source s to destination d, there is a corresponding data transfer from d back to s.

Link l_{sd} has a capacity constraint C_{sd}, which indicates the maximum number of requests that may be served in a time instant on that link. Note that we assume that all pieces of content are of equal size. Further, end-users are served by unicast flows, *i.e.*, servicing multiple requests of the same content will each require capacity on the link. The total capacity available at source s is given by $C_s = \sum_{\forall d} C_{sd}$. We further define the maximum available capacity over all sources as $C_{sM} = \max_s C_s$. The sum total capacity of the network is given by $C_{tot} = \sum_{sd} C_{sd}$. The number of links at source s is N_s, while the number of links at cache d is N_d.

Request Arrivals at Frontend Nodes

Under the switch abstraction of the CDN, we have request queues of size $q_s^c[k]$ at source s for content c at (discrete) time k. We denote the vector of all such queues (the system state) by $\vec{\mathcal{Q}}[k]$. The number of requests that arrive at time k is denoted $a_s^c[k]$. Arrivals are Bayesian, with finite mean λ_s^c and second moment η_s^c. We assume that for any $A \geq 0$, there exists a $\delta_A > 0$, such that $\mathbb{P}(a_s^c[k] \leq A : \forall c, s) > 1 - \delta_A \; \forall k$. Finally, we define $\Lambda \triangleq \max_{c,s} \lambda_s^c$. The arrival processes must satisfy the following conditions that are necessary for stability:

$$\sum_{\forall c} \lambda_s^c < \sum_{\forall d} C_{sd}. \tag{1}$$

Servicing Requests

We assume that sources are content-aware, in that they know what content is present in each cache that they have access to[1]. For simplicity of notation, we assume that each source is connected to all caches; all the analysis is valid even if this assumption does not hold. The presence of content c at cache d at time instant k is indicated by $p_d^c[k] \in \{0, 1\}$ with the vector of $p_d^c[k]$ denoted by $\vec{p}[k]$. For all k, the source requests for content from each cache is $\chi_{sd}^c[k] \in \{0, 1\}$, where,

$$\sum_{\forall c} \chi_{sd}^c(k) \leq 1 \; \forall (s, d) \text{ pairs}. \tag{2}$$

At most C_{sd} copies of the selected content c are then served by the cache. We denote the amount of *scheduled* service to a request queue $q_s^c[k]$, with respect to cache c as

$$\mu_{sd}^c[k] \triangleq \chi_{sd}^c[k] C_{sd}. \tag{3}$$

The total number of scheduled departures from request queue $q_s^c[k]$ over all caches is simply $\mu_s^c[k] \triangleq \sum_d \mu_{sd}^c[k]$. Since there are $q_s^c[k]$ requests for c at s, the number of copies of c that can be served is upper bounded by this value, and we refer to the *actual* number of departures that occur at s as,

$$\tilde{\mu}_s^c[k] \triangleq \min[\mu_s^c[k], q_s^c[k]]. \tag{4}$$

The evolution of the source queue containing requests for content c is then given by

$$q_s^c[k+1] \quad = \quad q_s^c[k] + a_s^c[k] - \tilde{\mu}_s^c[k] \tag{5}$$

Note that for all feasible schedules the departures must necessarily satisfy the capacity constraints,

$$\sum_{\forall c} \tilde{\mu}_s^c[k] \leq \sum_{\forall c} \mu_s^c[k] \leq C_s \tag{6}$$

Refreshing Cache Contents

Each cache d has a size B_d, which indicates the number of pieces of content that it can store. The cache size is likely to be much larger than the number of frontends that it serves, *i.e.* $B_d \geq N_d$. Again, for simplicity of notation we consider identical cache sizes B. The content present in the caches is refreshed periodically from the media vault with periodicity D. Our refresh model is that each source may request one item to be fetched from the media vault at time instants $k = nD$, where $n \in \mathbb{N}$. Thus, we have a regime in which

- At refresh instants sources may request any piece of content from each cache, since the chosen content would be fetched from the media vault, *i.e.*, for $k = nD$
$$\chi_{sd}^c[k] \text{ can be chosen as 1 independent of } p_d^c[k]. \tag{7}$$
- At inter-refresh instants sources may only request pieces of content that are currently present in the caches, *i.e.*, for $k \neq nD$, for each $\chi_{sd}^c[k] = 1$, we have,
$$\chi_{sd}^c[k] \times p_d^c[k] = 1 \tag{8}$$

Following terminology from switching literature, we will refer to the vector of requests made by sources as a *schedule,* $\vec{\chi}[k]$ and the policy for doing so as a *scheduling algorithm.*

Content Evictions

Finally, caches must evict certain pieces of content at the refresh instants to make room for the new content fetched. We denote eviction of content c at cache d at time k by $e_d^c[k] \in \{0, 1\}$. We will refer to the policy used for evictions as an *eviction algorithm.* Evictions must satisfy the following constraints,

- A requested item cannot be evicted:
$$\chi_{sd}^c[k] + e_d^c[k] \leq 1 \tag{9}$$
- Only content that is present can be evicted, *i.e.* for each $e_d^c[k] = 1$, we have,
$$e_d^c[k] \times p_d^c[k] = 1 \tag{10}$$

We will study the question of algorithm design in the next sections. Our objective is to develop algorithms that are *throughput optimal,* which means that as long as the arrival rates satisfy the necessary condition (1), the expected values of all the request queues will remain finite.

III. The Periodic Max-Weight Algorithm

Since our CDN model bears a resemblance to a switch, we are inspired by the Max-Weight scheduling algorithm that was shown to be throughput optimal for a switch [8]. Unlike a switch, however, we have to take content placement, eviction and scheduling decisions at refresh instants. Suppose that at refresh instants we find a schedule such that

$$\chi_{sd}^{c*}[k] = \arg\max_{\chi_{sd}^c} \sum_{\forall s,c,d} q_s^c[k] C_{sd} \chi_{sd}^c[k]. \tag{11}$$

We refer to the above optimization as Max-Weight optimization *independent* of cache contents (MWI). Note that the solution does not specify which pieces of content are to be evicted, and we could choose to simply evict a random subset of the cache contents that does not interfere with the schedule. Further, at the inter-refresh time instants, we require a schedule with the proviso that it only incorporates content that is already present in the caches. We could again use a Max-Weight schedule, except that it must now be calculated subject to the

[1]This assumption is based on private communication with Cisco Inc. on their latest CDN architecture.

presence of scheduled content (MWP). We refer to the policy that comprises of MWI at the refresh instants, and MWP at the inter-refresh instants as the Periodic Max-Weight scheduling algorithm (PMW). We define the policy in Algorithm 1.

Algorithm 1 Periodic Max-Weight Scheduling

At the refresh instants (MWI plus Evictions): For all $k = nD$, schedule the links subject to (2),

$$\vec{\chi}^*[k] := \arg\max_{\chi} \mathcal{W}_{\mathcal{I}}(\vec{\chi}[k], \vec{\mathcal{Q}}[k]), \tag{12}$$

where

$$\mathcal{W}_{\mathcal{I}}(\vec{\chi}[k], \vec{\mathcal{Q}}[k]) \triangleq \sum_{\forall s,c,d} q_s^c[k] C_{sd} \chi_{sd}^c[k] \tag{13}$$

Fetch the requested content from the media vault and, if needed, evict contents arbitrarily from the cache such that the conditions (9) and (10) are satisfied, *e.g.*, randomly select content that has not been scheduled to be evicted.

At the inter-refresh instants (MWP): For all $k \neq nD$, schedule the links subject to (2),

$$\vec{\hat{\chi}}[k] := \arg\max_{\chi} \mathcal{W}_{\mathcal{P}}(\vec{\chi}[k], \vec{\mathcal{Q}}[k], \vec{p}[k]), \tag{14}$$

where

$$\mathcal{W}_{\mathcal{P}}(\vec{\chi}[k], \vec{\mathcal{Q}}[k], \vec{p}[k]) \triangleq \sum_{\forall s,c,d} q_s^c[k] C_{sd} p_d^c[k] \chi_{sd}^c[k] \tag{15}$$

At $k = lD$, we define the weight of the schedule $w^*[k]$ as,

$$w^*[k] \triangleq \max_{\chi} \mathcal{W}_{\mathcal{I}}(\vec{\chi}[k], \vec{\mathcal{Q}}[k]). \tag{16}$$

At $k \neq lD$, we define the weight of the schedule $\hat{w}[k]$ as,

$$\hat{w}[k] \triangleq \max_{\chi} \mathcal{W}_{\mathcal{P}}(\vec{\chi}[k], \vec{\mathcal{Q}}[k], \vec{p}[k]). \tag{17}$$

Following (3), we refer to the scheduled service associated with $\vec{\chi}^*[k]$ as $\vec{\mu}^*[k]$, and similarly that with $\vec{\hat{\chi}}[k]$ as $\vec{\hat{\mu}}[k]$.

A. Stability and Performance Analysis of the PMW Algorithm

We will now show that the PMW algorithm stabilizes the system, and derive bounds on the total queue length under this policy. We first recall the Foster-Lyapunov stability criterion that will enable us to show such stability.

Theorem 1. *(Foster-Lyapunov stability criterion) Let* $\boldsymbol{Q}$ *be a countable state-space, and let* $\vec{\mathcal{Q}}[k]$ *be an irreducible, aperiodic, countable-state Markov chain. Suppose there exists a Lyapunov function* $V : \boldsymbol{Q} \to \mathbb{R}^+$*, and* C*, which is a finite subset of* $\boldsymbol{Q}$*. If* $\epsilon > 0$ *and* b *is a constant such that the drift*

$$\Delta V[k] = \mathbb{E}\left[V[k+1] - V[k] | \vec{\mathcal{Q}}[k]\right] \leq -\epsilon + b I_C,$$

then $\vec{\mathcal{Q}}[k]$ *is positive recurrent. Further, if* $g[k]$ *and* $f[k]$ *are two processes such that the drift can be expressed as*

$$\Delta V[k] \leq \mathbb{E}\left[g[k] - f[k] | \vec{\mathcal{Q}}[k]\right],$$

and the system is positive recurrent, then

$$\limsup_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} \mathbb{E}\left[f[i]\right] \leq \limsup_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} \mathbb{E}\left[g[i]\right]. \tag{18}$$

We first prove that the PMW algorithm is stable if the refresh periodicity $D = 1$, and the result will be used to show stability of the PMW policy with $D \in \mathbb{N}$. We have the following theorem.

Theorem 2. *The Periodic Max-Weight scheduling policy is throughput optimal for a refresh periodicity equal to one.*

Proof: Consider the Lyapunov function

$$V[k] = \frac{1}{2} \sum_{\forall c,s} \left(q_s^c[k]\right)^2. \tag{19}$$

The drift of the Lyapunov function is given by

$$\Delta V[k] \quad = \quad \mathbb{E}\left[V[k+1] - V[k] | \vec{\mathcal{Q}}[k]\right].$$

The drift can be simplified using (5), (6) and Jensen's inequality to yield[2],

$$\Delta V[k] \leq B_1 + B_2 + \sum_{c,s} \mathbb{E}\left[\left(q_s^c[k]\right)\left(\lambda_s^c - \mu_s^c[k]\right) | \vec{\mathcal{Q}}[k]\right] \tag{20}$$

where B_1 and B_2 are positive, bounded quantities given by

$$\begin{aligned} B_1 \quad &= \quad \frac{1}{2} \sum_{c,s} \mathbb{E}\left[a_s^{c2}[k] | \vec{\mathcal{Q}}[k]\right] + \frac{1}{2} \sum_{c,s} \mathbb{E}\left[\tilde{\mu}_s^{c2} | \vec{\mathcal{Q}}[k]\right] \\ &\qquad - \sum_{c,s} \mathbb{E}\left[\lambda_s^c \tilde{\mu}_s^c[k] | \vec{\mathcal{Q}}[k]\right], \end{aligned} \tag{21}$$

$$\begin{aligned} B_2 \quad &= \quad \min\left[\sum_{c,s} C_s^2, \sum_s \mathbb{E}\left[C_s \tilde{\mu}_s[k] | \vec{\mathcal{Q}}[k]\right]\right] \\ &\qquad - \sum_{c,s} \mathbb{E}\left[\tilde{\mu}_s^{c2}[k] | \vec{\mathcal{Q}}[k]\right]. \end{aligned} \tag{22}$$

We use a similar line of reasoning that was used earlier in [8] and [14] in order to bound the drift. Consider a vector $\vec{\epsilon}$ with each of its elements equal to a constant $\epsilon > 0$ such that $\left(\vec{\lambda} + \vec{\epsilon}\right)$ lies in the capacity region (1). Let $\mathcal{M} = \{\vec{\mu}_i\}$ be the set of all feasible service schedules. The convex combination of all feasible service schedules $\sum_i \beta_i \vec{\mu}_i$ defines the capacity region, where we have $\sum_i \beta_i = 1$. It then follows that $\vec{\lambda} + \vec{\epsilon} \leq \sum_i \beta_i \vec{\mu}_i$ for some β. The scheduled service at time k also satisfies $\mu^*[k] \leq \sum_i \beta_j \vec{\mu}_j$. We then deduce from (3) and (12) that

$$\sum_{\forall s,c} \mathbb{E}\left[q_s^c \left(\lambda_s^c - \mu_s^{c*}[k]\right) | \vec{\mathcal{Q}}[k]\right] \leq -\epsilon \sum_{\forall s,c} q_s^c[k]. \tag{23}$$

Examining the drift in (20) when $\mu_s^c[k] = \mu_s^{c*}[k]$,

$$\begin{aligned} \Delta V \quad &\leq \quad B_1 + B_2 \\ &\qquad + \sum_{c,s} \mathbb{E}\left[\left(q_s^c[k]\right)\left(\lambda_s^c - \mu_s^{c*}\right) | \vec{\mathcal{Q}}[k]\right] \\ &\overset{(a)}{\leq} \quad B_1 + B_2 - \epsilon \sum_{c,s} q_s^c[k], \end{aligned}$$

[2]Please see details in [15].

where (a) follows from (23). Hence, except for a finite subset of **Q**, the drift is negative for the PMW policy with an eviction periodicity equal to 1, and the proof follows by using the Foster-Lyapunov criterion. ■

We now prove that the PMW algorithm stabilizes the system for any finite refresh periodicity D.

Theorem 3. *The Periodic Max-Weight scheduling policy is throughput optimal for all finite refresh periodicities.*

Proof: From Theorem 2 we already know that at refresh instants, the drift of the Lyapunov function given in (19) satisfies the Foster-Lyapunov criterion. Hence, we only need to consider the drift at inter-refresh instants. When $k \neq nD$, the drift simplifies to (20) with $\mu_s^c[k] = \hat{\mu}_s^c[k]$ yielding,

$$\Delta V \leq B_1 + B_2 + \sum_{c,s} \mathbb{E}\left[\left(q_s^c[k]\right)\left(\lambda_s^c - \hat{\mu}_s^c[k]\right) | \vec{\mathcal{Q}}[k] \right]. \quad (24)$$

Let $l \triangleq \max\{n : nD < k\}$, *i.e.,* l is the prior refresh instant nearest to k. We have the following two useful relations:

- Since the maximum number of departures from the system is upper bounded at any time instant,
$$\begin{aligned} q_s^c[k] &\geq q_s^c[k-1] - C_{sM} \\ \Rightarrow q_s^c[k] &\geq q_s^c[lD] - C_{sM}(k - lD + 1). \end{aligned} \quad (25)$$
- ince the maximum number of arrivals to all queues in the system is upper bounded by $A \geq 0$ with probability $(1 - \ \delta_A)$ at any time instant,
$$q_s^c[k-1] \geq q_s^c[k] - A$$
Hence with probability $\Delta_A = (1 - \ \delta_A)^{(k-lD)}$, we have,
$$q_s^c[lD] \geq q_s^c[k] - A(k - lD). \quad (26)$$

We then have with probability Δ_A,

$$\begin{aligned} \sum_{c,s,d} \hat{\mu}_{sd}^c[k] q_s^c[k] &\overset{(a)}{\geq} \sum_{c,s,d} \mu_{sd}^{c*}[lD] q_s^c[k] \\ &\overset{(b)}{\geq} \sum_{c,s,d} \mu_{sd}^{c*}[lD] q_s^c[lD] - C_{tot} C_{sM}(k - lD + 1) \\ &\overset{(c)}{\geq} \sum_{c,s,d} \mu_{sd}^{c*}[k] q_s^c[lD] - C_{tot} C_{sM}(k - lD + 1) \\ &\overset{(d)}{\geq} \sum_{c,s,d} \mu_{sd}^{c*}[k] q_s^c[k] - C_{tot} C_{sM}(k - lD + 1) \\ &\quad - C_{tot} A(k - lD), \end{aligned}$$

where (a) and (c) follow directly from the fact that the optimized solution at any instant has a higher weight than any other schedule, (b) follows from (25), and (d) from (26). Thus, we have just shown that with probability Δ_A,

$$-\sum_{c,s} \hat{\mu}_s^c[k] q_s^c[k] \leq -\sum_{c,s} \mu_s^{c*}[k] q_s^c[k] + f(D), \quad (27)$$

where

$$f(D) = C_{tot} C_{sM}(k - lD + 1) + C_{tot} A(k - lD). \quad (28)$$

Now, from (24) and (27) we obtain,

$$\begin{aligned} \Delta V &\leq B_1 + B_2 + \Delta_A f(D) \\ &\quad + \Delta_A \sum_{c,s} \mathbb{E}\left[\left(q_s^c[k]\right)\left(\lambda_s^c - \mu_s^{c*}[k]\right) | \vec{\mathcal{Q}}[k] \right] \\ &\quad + (1 - \Delta_A) \sum_{c,s} \mathbb{E}\left[\left(q_s^c[k]\right)\left(\lambda_s^c - \hat{\mu}_s^c[k]\right) | \vec{\mathcal{Q}}[k] \right] \\ &\overset{(e)}{\leq} B_1 + B_2 + \Delta_A f(D) \\ &\quad + \left((1 - \Delta_A)\Lambda - \Delta_A \epsilon\right) \sum_{c,s} q_s^c[k] \\ &\leq B_1 + B_2 + \Delta_A f(D) \\ &\quad - \alpha_A \sum_{c,s} q_s^c[k], \end{aligned} \quad (29)$$

where

$$\alpha_A \triangleq \left(-(1 - \Delta_A)\Lambda + \Delta_A \epsilon\right).$$

(e) arises from (23) and from the fact that,

$$\sum_{c,s} \mathbb{E}\left[\left(q_s^c[k]\right)\left(\lambda_s^c - \hat{\mu}_s^c[k]\right) | \vec{\mathcal{Q}}[k] \right] \leq \Lambda \sum_{c,s} q_s^c[k].$$

By selecting A in such a way that $\alpha_A > 0$, the drift is negative except for a finite subset of **Q**. Therefore, combining the results of Theorem 2 and the relation (29), the Periodic Max-Weight Scheduling Algorithm is stable for all finite refresh periodicities. ■

We just proved that the PMW Algorithm stabilizes the system when the periodicity is finite. However, stability only guarantees the finiteness of the queues, but does not yield information about delays. From Little's Law, the sum of the queue lengths in the system is related to the delays, and we now find upper bounds on the expected queue lengths.

Corollary 4. *The sum of the queue backlogs in the system using the PMW Algorithm with unit refresh periodicity satisfies*

$$\begin{aligned} \sum_{s,c} \mathbb{E}\left[q_s^c[k]\right] &\leq \frac{\sum_{s,c} \eta_s^c}{2\epsilon} - \frac{3\sum_{s,c} \lambda_s^{c2}}{2\epsilon} \\ &\quad + \frac{\min\left[\sum_{c,s} C_s^2, \sum_s C_s \lambda_s\right]}{\epsilon}. \end{aligned}$$

Proof: We use the steady state condition of the Markov chain given in (18) to find the expected queue lengths. From Theorem 1 and (18) we have,

$$0 \leq B_1 + B_2 - \epsilon \sum_{c,s} \mathbb{E}\left[q_s^c[k]\right].$$

Since the algorithm is stable,

$$\mathbb{E}\left[\tilde{\mu}_s^c[k]\right] = \lambda_s^c \ \forall c, s,$$

and we make use of this fact in the expansions of B_1 and B_2

defined in (22) and (21). Thus,

$$\begin{aligned}0 &\leq \frac{1}{2}\sum_{c,s}\mathbb{E}\left[a_s^{c2}[k]\right]-\frac{1}{2}\sum_{c,s}\mathbb{E}\left[\tilde{\mu}_s^{c2}\right]\\ &\quad -\sum_{c,s}\lambda_s^{c2}+\min\left[\sum_{c,s}C_s^2,\sum_s C_s\lambda_s\right]\\ &\quad -\epsilon\sum_{c,s}\mathbb{E}\left[q_s^c[k]\right]\\ &\overset{(a)}{\leq} \frac{1}{2}\sum_{c,s}\eta_s^c-\frac{3}{2}\sum_{c,s}\lambda_s^{c2}+\min\left[\sum_{c,s}C_s^2,\sum_s C_s\lambda_s\right]\\ &\quad -\epsilon\sum_{c,s}\mathbb{E}\left[q_s^c[k]\right],\end{aligned}$$

where (a) arises from Jensen's inequality. The proof follows. ■

The above result characterizes the rate at which the sum of the queue lengths increases if the number of content files served by the CDN increases, with no proportionate increase in the delivery capacity. We now find a similar bound on the expected queue lengths for the PMW Algorithm with a refresh periodicity greater than one.

Corollary 5. *The sum of the queue lengths for the CDN that uses the PMW Algorithm with period D satisfies*

$$\begin{aligned}\sum_{c,s}\mathbb{E}\left[q_s^c[k]\right] &\leq \frac{\sum_{s,c}\eta_s^c}{2\alpha_A}-\frac{3\sum_{s,c}\lambda_s^{c2}}{2\alpha_A}\\ &\quad +\frac{\min\left[\sum_{c,s}C_s^2,\sum_s C_s\lambda_s\right]}{\alpha_A}\\ &\quad +\frac{\Delta_A f(D)}{\alpha_A}\end{aligned}$$

where $f(D)$ is given by (28).

Proof: The proof follows from (29) in the same manner as Corollary 4. ■

From (28) it is clear that as D increases, the difference $k-lD$ increases on average, with an upper bound of $D-1$. Thus, on average, a larger periodicity D corresponds to a larger value of $f(D)$, which, from Corollary 5, implies an increase in average queue length. The result is intuitive since a longer refresh interval implies a greater propensity for the cache contents to become stale. The analytical characterization indicates that an increase in refresh periodicity lengthens delays in a proportional manner.

Since links from sources to caches do not interfere with each other, the PMW policy simplifies to a "longest queue first" (LQF) schedule, which can be solved independently at each source node in a distributed manner. At the caches we simply evict a random subset of unscheduled content files to create space for the scheduled ones. Hence, the complexity of the algorithm is low.

IV. Exploring eviction policies

It is intuitively clear that evictions have an impact on performance. For example, if content items that have large request queue lengths are evicted, they cannot be served during the inter-refresh period, to the detriment of user delay. There might be eviction policies that result in short queue lengths, while maintaining system stability.

Consider any two policies R and M, both of which are known to be throughput optimal. Suppose that M is such that the weight $w^M[k]\geq w^R[k]$, *i.e.,* the schedule it selects always has a greater weight than the one selected by R. Then

$$\sum_{c,s,d}C_{sd}q_s^c[k]\chi_{sd}^{cM}[k] \geq \sum_{c,s,d}C_{sd}q_s^c[k]\chi_{sd}^{cR}[k]$$

$$\sum_{c,s}q_s^c[k]\mu_s^{cM}[k] \geq \sum_{c,s}q_s^c[k]\mu_s^{cR}[k] \quad (30)$$

$$\Rightarrow\sum_{c,s}q_s^c[k]\left(\lambda_s^c-\mu_s^{cM}[k]\right) \leq \sum_{c,s}q_s^c[k]\left(\lambda_s^c-\mu_s^{cR}[k]\right) \quad (31)$$

Hence, using (31) and from the expression for the drift in (20), we deduce that except for a finite subset of $\mathbf{Q}$,

$$\Delta V^M\leq\Delta V^R\leq 0$$

and it follows from an argument similar to Corollaries 4 and 5 that the upper bound on the queue lengths under M is smaller than that under R. Thus, a greater weight of the schedule at each time instant could result in a shorter queue length. While the MWI schedule indeed does maximize this weight at the refresh times, the evictions performed at refresh instants would impact the space over which MWP is calculated during the inter-refresh times. In other words, appropriate evictions during refresh instants could result in a greater negative drift during inter-refresh instants.

The fetch and eviction decisions made at refresh time lD impact the availability of content at some time k. Define the presence of content at a cache $p_d^c[k]$ as follows:

$$p_d^c[k]=\left(1-e_d^c[lD]\right)\left(p_d^c[lD]\right)+\left(1-p_d^c[lD]\right)\left(X_d^{c*}[lD]\right), \quad (32)$$

where

$$X_d^{c*}[lD]=\begin{cases}1 & \text{if } \sum_s\chi_{sd}^{c*}[lD]\geq 1\\ 0 & \text{else.}\end{cases}$$

Consider the two policies M and R, both of which employ the PMW policy for scheduling and differ only in their evictions. Let R correspond to random evictions. Denote the eviction variables as $e_d^{cR}[lD]$ and $e_d^{cM}[lD]$ for the two policies. In order for M to perform better than R, from (30) we would like,

$$\sum_{c,s}q_s^c[k]\hat{\mu}_s^{cM}[k]\geq\sum_{c,s}q_s^c[k]\hat{\mu}_s^{cR}[k],$$

to hold good.

From (14) and since the second term in (32) is the same for both policies,

$$\begin{aligned}&\Rightarrow\sum_{s,d}C_{sd}\sum_c q_s^c[k]\left(1-e_d^{cM}[lD]\right)\left(p_d^c[lD]\right)\\ &\geq\sum_{s,d}C_{sd}\sum_c q_s^c[k]\left(1-e_d^{cR}[lD]\right)\left(p_d^c[lD]\right)\end{aligned}$$

If exactly the same number of arrivals took place for all queues in the interval $[lD, k]$, then we could ensure the above condition holds by choosing to evict,

$$e_d^{c*}[k] = \arg\min_{e_d^{c*}} \sum_c \left(\sum_s (q_s^c[k] p_d^c[k] C_{sd}) \right) e_d^c[k] \ \forall d \ \forall k$$

subject to (9). In other words, we propose a *Min-Weight Eviction* strategy to complement the Max-Weight (Independent) Algorithm that is used at refresh times. More formally, we have the following algorithm.

Algorithm 2 PMW Scheduling with Min-Weight Evictions

```
    **At the refresh instants**, for all $k = lD$, schedule based on MWI as in Algorithm 1. Evict contents based on,
    $e_d^{c*}[k] := \arg\min_{e_d^{c*}} \sum_c \left( \sum_s (q_s^c[k] p_d^c[k] C_{sd}) \right) e_d^c[k] \ \forall d \ \forall k$
    subject to condition (9).
    **At the inter-refresh instants**, for all $k \neq lD$, schedule based on MWP as in Algorithm 1.
```

We then have the following straightforward theorem.

Theorem 6. *The PMW Algorithm with Min-Weight Evictions is throughput optimal for all finite refresh periodicities.*

Proof: It follows from Theorem 3 that since the PMW algorithm is stable for *any* feasible eviction policy, it is also throughput-optimal for the Min-Weight Evictions policy for all finite refresh periodicities. ■

V. Exploring feasible scheduling strategies Iterative Periodic Max-Weight Algorithm

An important observation on LQF scheduling in the context of wireless networks in [16] is that it could result in wastage if a queue scheduled for service does not have enough packets to utilize the entire capacity. They then propose an iterative solution to lessen such wastage. In our context, we attempt a similar scheme to ensure non-zero service on all the links at all instants. We propose the IPMW - Iterative Periodic Max-Weight Scheduling policy which uses the Iterative variants of MWI and MWP (IMWI / IMWP).

We introduce some additional terminologies for our new policy. Let us refer to the schedules as $\chi_{sd}^{cI*}[k]$ / $\mu_{sd}^{cI*}[k]$ for $k = lD$ and $\hat{\chi}_{sd}^{cI}[k]$ / $\hat{\mu}_{sd}^{cI}[k]$ for $k \neq lD$. We refer to d as the cache whose link is being scheduled in the current iteration. $\mathcal{E}^s$ is the set of all caches ordered in the descending capacities of their links to that node s, *i.e.*, $C_{sd} \leq C_{s(d-1)} \ \forall d \in \mathcal{E}^s$. $\mathcal{E}_{PERM}^s$ is set of all permuted cache orderings and $\mathcal{E}_j^s \in \mathcal{E}_{PERM}^s \ \forall 1 \leq j \leq E!$. $\mathcal{F}^s[k]$ is the set of all caches that serviced the selected queue until the current iteration. The algorithm is described as follows.

Theorem 7. *The Iterative Periodic Max-Weight scheduling policy is throughput optimal, for all finite refresh periodicities.*

Algorithm 3 Iterative Periodic Max-Weight Scheduling

```
**At the refresh instants**, for all $k = lD$, at each source $s$,
**repeat**
    Find the longest queue $q_s^{cI*}[k]$.
    **for** each $d \in \mathcal{E}^s \backslash \mathcal{F}^s[k]$ (in order), update the queue and schedule the link as, **do**
        $q_s^{cI*}[k] := \left(q_s^{cI*}[k] - C_{sd}\right)^+,$
        $\chi_{sd}^{cI*}[k] := 1,$
        $\mathcal{F}^s[k] := \mathcal{F}^s[k] \cup \{d\}.$
        Fetch the missing content and evict arbitrarily subject to (9) and (10).
        **if** $q_s^{cI*}[k] = 0$, **then**
            break.
        **end if**
    **end for**
**until** $\mathcal{F}^s[k] = \mathcal{E}^s$.
**At the inter-refresh instants**, for all $k \neq lD$, at each source $s$, we initially estimate schedules, queue updates, system throughputs without actual implementation as,
**for** each $\mathcal{E}_j^s \in \mathcal{E}_{PERM}^s$, **do**
    **Step 1** Find the MWP schedule $\vec{\hat{\chi}}[k]$.
    **for** each $d \in \mathcal{E}_j^s$ (in order), estimate the throughput $\mu_{sj}^e[k]$, update the queue and schedule the link as, **do**
        **for** each $c$, **do**
            **if** $q_{sj}^c[k]\hat{\chi}_{sd}^c[k] = 1$, **then**
                $\mu_{sj}^e[k] := \mu_{sj}^e[k] + \min[C_{sd}, q_{sj}^c[k]],$
                $q_{sj}^c[k] := \left(q_{sj}^c[k] - C_{sd}\right)^+,$
                $\chi_{sdj}^{ce}[k] := 1,$
                $\mathcal{F}_j^{se}[k] := \mathcal{F}_j^{se}[k] \cup \{d\}.$
            **end if**
        **end for**
    **end for**
    **Step 2**
    **for** each $d \in \mathcal{E}_j^s \backslash \mathcal{F}_j^{se}[k]$ (in order), **do**
        Find the longest queue $q_{sdj}^{ce}[k]$ subject to (8) for that cache $d$. Estimate the throughput, update the queue and schedule the link as,
        $\mu_{sj}^e[k] := \mu_{sj}^e[k] + \min[C_{sd}, q_{sdj}^{ce}[k]],$
        $q_{sdj}^{ce}[k] := \left(q_{sdj}^{ce}[k] - C_{sd}\right)^+,$
        $\chi_{sdj}^{ce}[k] := 1,$
        $\mathcal{F}_j^{se}[k] := \mathcal{F}_j^{se}[k] \cup \{d\}.$
    **end for**
    Compute the net estimated throughput for this cache ordering as $\mu_j^e[k] := \sum_s \mu_{sj}^e[k]$.
**end for**
```

Now find $\arg\max_j \mu^e_j[k]$ and use the schedule $\vec{\chi^e}_j[k]$ corresponding to this optimal link ordering as $\hat{\vec{\chi}}^I[k]$.

The average queue length is no greater than that of the Periodic Max-Weight scheduling policy.

Proof: The proof follows directly from that of Theorem 3 since at least exactly the same departures happen in IPMW as in the PMW Algorithm. ∎

Finally, corresponding to Algorithm 2, we have a version of IPMW coupled with min-weight evictions, which is also throughput optimal, and which has an average queue length at most that of PMW with min-weight evictions.

Algorithm 4 IPMW Scheduling with Min-Weight Evictions

At the refresh instants, for all $k = lD$, schedule based on IMWI as in Algorithm 3. Evict contents based on,

$$e^{c*}_d[k] := \arg\min_{e^{c*}_d} \sum_c \left(\sum_s (q^c_s[k] p^c_d[k] C_{sd}) \right) e^c_d[k] \ \forall d \ \forall k$$

subject to condition (9).

At the inter-refresh instants, for all $k \neq lD$, schedule based on IMWP as in Algorithm 3.

VI. Simulations

We illustrate the insights from our analytical model by simulating our CDN abstraction using C++. In our simulation, each source node has a different Zipf rank distribution over content [17], [18], and arrivals take place proportional to these ranks. We are interested in the following scenarios:

1) A fully connected CDN, with $S = 7$, $E = 2$, $C = 16$ and $B = 10$ with link capacities chosen arbitrarily. Loads were chosen within 1% of the link capacities. We derived our analytical results for this topology, although they are easily generalized to arbitrary topologies.
2) A CDN topology that follows the Abilene network [19], illustrated in Figure 2. Sources can access a local cache, as well as other neighboring caches. The objective is to consider an arbitrary partially connected topology. Links are capacitated, and we have $S = 11, E = 11, C = 16$. Loads were chosen within 1% of the link capacities. We expect all our analytical insights to also apply here.
3) The Abilene network in which refresh periodicities are different at different caches. We set the baseline refresh periods for each of the caches D_d and then vary them by common multiples to view the variation of the performance with the entire set of D_d for all policies.

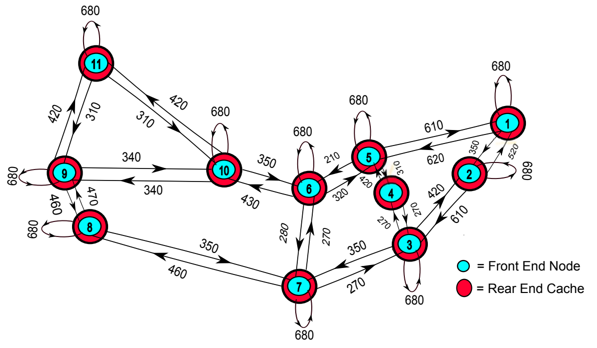


Fig. 2. The Abilene Network Topology where each node represents a source and an associated cache. Sources may route requests to neighboring caches.

Variation of delay with eviction periodicity

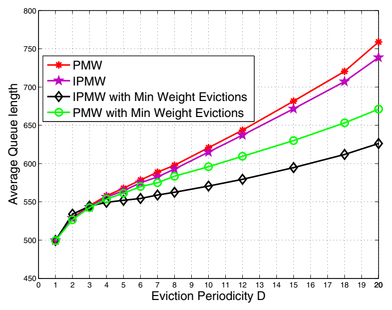
We conduct a first set of simulations to illustrate the performance of our routing-placement-eviction policies as the refresh periodicity increases. In earlier sections we had tied large (negative) Lyapunov drifts with shorter queue lengths, and tried to design algorithms that would achieve such drifts. We now show that such design is indeed valid from a performance standpoint. The results are shown in Figure 3. We see that the PMW policy has the largest queue lengths, which corresponds to its greatest Lyapunov drift. The PMW policy with Min-weight evictions performs better, as expected by our maximizing the cache weight through appropriate evictions. The iterative versions of both algorithms outperform the non-iterative ones; again the result follows from Lyapunov drift arguments.

Variation of Delays with Cache Size

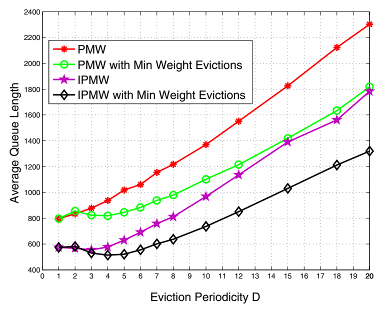
We now explore performance variation with the cache size. Intuition suggests that the decision to cache the “useful” objects and the eviction technique employed play an important role only when $B > N_d$,. We studied this facet of the problem using the Abilene network with homogeneous eviction periods, and varied the cache size as $B \in \{5, 7, 10\}$ along with D. In Figure 4(c) we observe that the queue lengths for all the policies are longer for smaller caches. Also the delays grow large with D for $B = 5$, while they remain almost independent of refresh periodicity for caches as large as $B(=10) >> \max[N_d](=4)$. This is understandable since the range of content that is available at cache to schedule during the refresh periods is more diverse for larger caches. For the very same reason, the advantage offered by the Min-Weight eviction policies in terms of performance is considerable only for small cache sizes of $B = 5$. This gain reduces through $B = 7$ and the eviction policy seems to lose its relevance for $B = 10$, where the random policy converges in performance with its Min-Weight Eviction counterpart.

VII. Conclusion

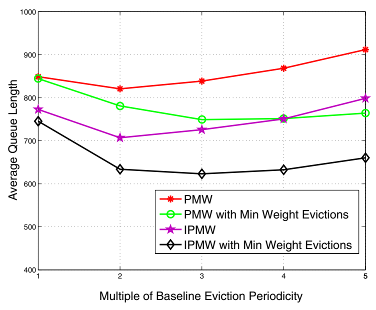
In this paper we studied algorithms for request routing, content placement, and content eviction in content distribution networks. We used the abstraction of a switch to model the CDN, and our objective was to design algorithms that would be throughput optimal (stabilize the system) as well as yield



(a) Fully Connected Topology

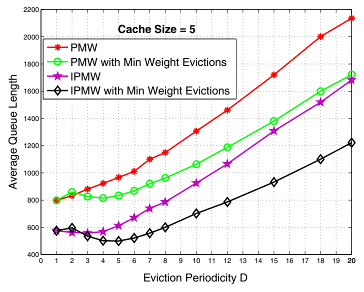


(b) Abilene Network Topology

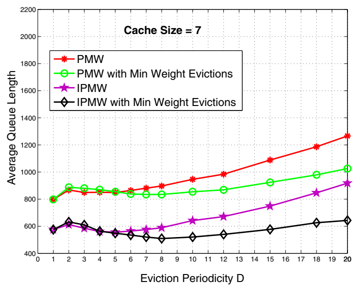


(c) Heterogeneous Eviction Periodicities

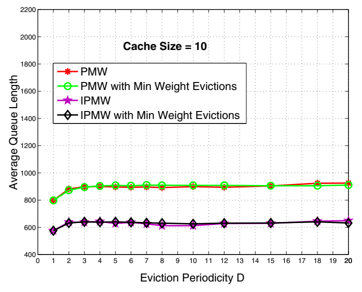
Fig. 3. Variation of average queue length with refresh periodicity for different algorithms.



(a) Cache Size $B = 5$



(b) Cache Size $B = 7$



(c) Cache Size $B = 10$

Fig. 4. Variation of average queue lengths with eviction periodicity for variable cache sizes.

short queues. Our main constraints were finite cache sizes and the periodicities with which content is refreshed in the caches. We showed how algorithms that engender large, negative Lyapunov drifts in the system are desirable, since such drifts beget short average queue lengths. We developed two algorithms—one with random evictions, and one with Min-Weight evictions, and illustrated the superior potential of the latter in causing such drifts. We also created iterative versions of both algorithms that are more efficient, and hence cause still shorter queue lengths. Our current work only accounted for requests with a soft delay tolerance. Future work includes streaming traffic with requests that have hard delay constraints, and which are dropped if such a constraint cannot be met.

References

[1] C. Labovitz, D. McPherson, and S. Iekel-Johnson, "Atlas Internet Observatory 2009 annual report," in *Proc. NANOG-47*, MI, Jun. 2009.

[2] R. M. P. Raghavan, *Randomized Algorithms*. New York: Cambridge University Press, 1995.

[3] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," in *Proc. 1997 USENIX Symposium on Internet Technology and Systems*, Berkeley, CA, Dec. 1997.

[4] K. Psounis and B. Prabhakar, "Efficient randomized web-cache replacement schemes using samples from past eviction times," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 441–455, Nov. 2002.

[5] N. Laoutaris, O. Telelis, Orestis, V. Zissimopoulos, and I. Stavrakakis, "Distributed selfish replication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 12, pp. 1401–1413, Dec. 2006.

[6] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. IEEE INFOCOM 2010*, San Diego, CA, Mar. 2010.

[7] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, Dec. 1992.

[8] ——, "Dynamic server allocation to parallel queues with randomly varying connectivity," *IEEE Transactions on Information Theory*, vol. 39, no. 2, pp. 466–478, Mar. 1993.

[9] X. Lin and N. Shroff, "Joint rate control and scheduling in multihop wireless networks," in *Proc. 43rd IEEE Conference on Decision and Control (CDC 2004)*, Paradise Islands, Bahamas, Dec. 2004.

[10] A. Stolyar, "Maximizing queueing network utility subject to stability: Greedy primal-dual algorithm," *Queueing Syst. Theory Appl.*, vol. 50, no. 4, pp. 401–457, 2005.

[11] A. Eryilmaz and R. Srikant, "Joint congestion control, routing, and mac for stability and fairness in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1514–1524, Aug. 2006.

[12] M. Neely, E. Modiano, and C.-P. Li, "Fairness and optimal stochastic control for heterogeneous networks," *IEEE/ACM Transactions on Networking*, vol. 16, no. 2, pp. 396–409, Apr. 2008.

[13] M. Neely, "Delay analysis for max weight opportunistic scheduling in wireless systems," *IEEE Transactions on Automatic Control*, vol. 54, no. 9, pp. 2137–2150, Sept. 2009.

[14] L. Le, K. Jagannathan, and E. Modiano, "Delay analysis of maximum weight scheduling in wireless ad hoc networks," in *Proc. 43rd Annual Conference on Information Sciences and Systems (CISS 2009)*, Baltimore, MD, Mar. 2009.

[15] M. M. Amble, P. Parag, S. Shakkottai, and L. Ying, "Content-aware caching and traffic management in content distribution networks," tech. Report available at http://www.ece.tamu.edu/~sshakkot/cachingtech.pdf.

[16] S. Bodas, S. Shakkottai, L. Ying, and R. Srikant, "Scheduling in multichannel wireless networks: rate function optimality in the small-buffer regime," in *Proc. ACM SIGMETRICS'09*, Jun. 2009.

[17] L. Breslau, P. Cue, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. IEEE INFOCOM 1999*, New York, Mar. 1999.

[18] S. Glassman, "A caching relay for the world wide web," *Computer Networks and ISDN Systems*, vol. 27, no. 2, pp. 165–173, 1994.

[19] "Abilene network. detailed information about the objectives, organization, and development of the abilene network," Internet2, 2005. [Online]. Available: www.internet2.edu/pubs/200502-IS-AN.pdf