CONTENT-AWARE CACHING AND TRAFFIC MANAGEMENT IN CONTENT DISTRIBUTION NETWORKS

A Thesis

by

MEGHANA MUKUND AMBLE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2010

Major Subject: Electrical Engineering

CONTENT-AWARE CACHING AND TRAFFIC MANAGEMENT IN CONTENT DISTRIBUTION NETWORKS

A Thesis

by

MEGHANA MUKUND AMBLE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Srinivas Shakkottai
Committee Members,	Narasimha Reddy
	Natarajan Gautam
	Jean-Francois Chamberland-Tremblay
Head of Department,	Costas N. Georghiades

December 2010

Major Subject: Electrical Engineering

ABSTRACT

Content-aware Caching and Traffic Management in Content Distribution Networks.

(December 2010)

Meghana Mukund Amble, B.Tech, National Institute of Technology, Karnataka, India

Chair of Advisory Committee: Dr. Srinivas Shakkottai

The rapid increase of content delivery over the Internet has lead 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. Our objective in this work is to design feasible algorithms that solve this trio of problems.

We abstract the system of front-end source nodes and back-end 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 content. For each request that is routed to a cache, a corresponding data file is transmitted back to the source across links of finite capacity. Caches are of finite size, and the content of the caches can be refreshed periodically. A requested but missing item is fetched to the cache from the media vault of the CDN. In case of a lack of adequate space at the cache, an existing, unrequested item may be evicted from the cache in order to accommodate a *new* item. Every such cache refresh or media vault access incurs a finite cost. Hence the refresh periodicity allowed to the system represents our system cost. In order to obtain small user delays, our algorithms must consider the lengths of the request queues that build up at the nodes. 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 using instantaneous system state information. The design yields insight into the impact of different cache refresh and eviction policies on queue length. We use this and construct throughput optimal algorithms that engender short queue lengths. We then propose a regime of algorithms which remedies the inherent problem of wastage of capacity. We also develop heuristic variants, and we study their performance.

We illustrate the potential of our approach and validate all our claims and results through simulations on different CDN topologies.

To my Parents

ACKNOWLEDGMENTS

I guess the person whom I really need to thank would be someone whose creative ideas, and guidance in every way, allowed me to realize this thesis - Dr. Srinivas Shakkottai. I am not only grateful for the official role of my research adviser that he played to perfection, but also for the person that he is, which made my research experience all the more interesting, rewarding, memorable and enjoyable. I would also like to thank Dr.Narasimha Reddy, Dr.Natarajan Gautam and Dr. Jean-Francois Chamberland-Tremblay for willing to be on my committee and providing me with their support. I would like to thank the couple, without whom, I would not be here pursuing my graduate studies - my parents. They are the one constancy I have relied upon through life, for their unfailing love and encouragement. On a final note, being the theist that I am, I acknowledge God, for whom any number of words will seem insufficient.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 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 for 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 suffer 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.

The journal model is *IEEE/ACM Transactions on Networking*.

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.

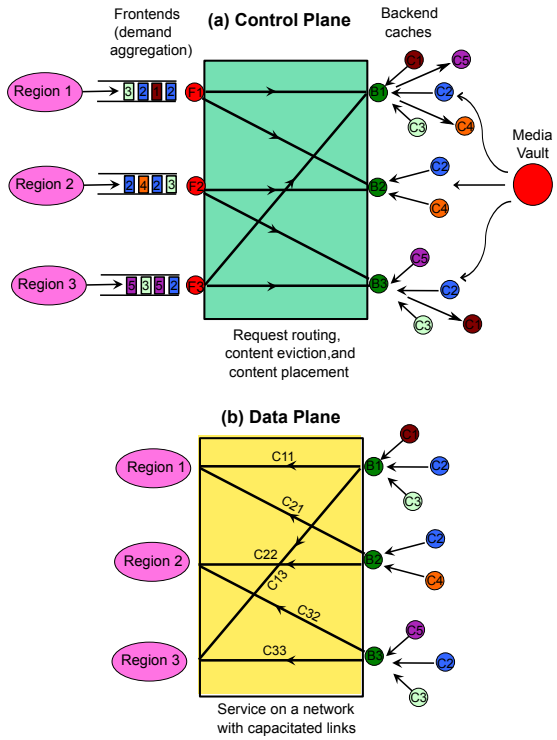


Fig. 1. A content distribution network. (a) Control plane: Requests arrive at front-end servers (S), and must be routed to one of (possibly) several back-end caches (D) that have the content. Caches can only host a finite number of content files (C), and the caches may be refreshed by placement and eviction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

In this work, 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. We then provide quantitative performance analysis that could further guide the design of distributed CDNs. This work is in submission to IEEE Infocom 2011.

At the refresh instants, the caches are allowed to provide service to any content type, while at the inter-refresh instants, service is provided subject to presence of content at the cache.

Our initial regime of algorithms uses instantaneous queue length information in the system to solve the problem. This approach discussed in Chapter IV, follows naturally from trying to stabilize a system, using a Lyapunov function that is quadratic

in the request queue lengths. We use Foster Lyapunov techniques to establish that the algorithms are throughput-optimal and then determine the upper bounds on the delays in the system. Our initial algorithm provides the solution for request routing-service at all times, handling evictions arbitrarily at the refresh instants. We shall refer to such a technique as a two-step approach in this work.

In Chapter V, we introduce the problem of evictions at the caches. We relate the average queue size to the drift of the Lyapunov function, 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 then use this insight to design an eviction technique that uses queue length information for making eviction decisions. We formulate a distributed, low-complexity, two-step policy that incorporates our initial throughput-optimal scheduling strategy, and this new eviction technique. Finally, utilizing the nature of scheduling and eviction decisions made by this two-step policy, we attempt to explore a heuristic, joint-scheduling-and-eviction algorithm.

Our initial regime suffers from an inherent drawback in that it can potentially create residual, unused capacity on the links. This problem has been studied earlier in [3] in the context of wireless down-link scheduling. In Chapter VI, we therefore design another class of algorithms, which still uses the queue information, but in a manner so as to ensure at least non-zero service on all the links, while guaranteeing the same amount of service that occurred via our basic algorithm. We refer to this as the iterative regime, as opposed to the initial non-iterative class. The state information here is immediately updated and scheduling is performed iteratively. We devise a throughput-optimal algorithm in this domain, that provably yields lower drifts than our non-iterative, two-step policy. Continuing along the same lines, we develop a heuristic scheme that intuitively should out-perform our first iterative algorithm, as

it guarantees complete capacity usage, instead of just non-zero service on the links.

In Chapter VII, we implement our various content placement, routing and eviction schemes in simulations of the CDN with different network topologies, and observe their performance. We then attempt to validate our results by studying the delays in the system for all scenarios, as the refresh periodicity and the cache size constraints are varied. We confirm that the delays in the system increase with the eviction costs for small cache sizes in the network, irrespective of the scheduling policy employed. The iterative algorithms show considerably lower queue lengths than the non-iterative ones. Our ideas on the performance of the heuristic schemes are validated. Finally, we observe that for any scheduling policy, the eviction strategy that uses queue length information out-performs its random counterpart, for large eviction costs and small cache size regimes.

CHAPTER II

RELATED WORK

Considering the nature of our problem and the solution framework that we are attempting to explore, we felt the need to review two vastly diverse areas of research.

The classical problem of caching and eviction has been visited in the context of memory caches, On-Line Web Caching at HTTP Proxies and distributed Web Storage systems. For example, [2,4,5] focus on on-line deterministic eviction algorithms (LRU, FIFO and LFU) and its variants such as greedy dual size and randomized versions. This kind of analysis involves the use of malicious adversaries posing worst case request scenarios to the system, and then comparing performance with the optimal off-line algorithm. The cost metric for a request sequence is the number of misses incurred. This is then used in evaluating the competitive coefficient which creates a standard of comparison amongst algorithms. LRU is the most popular eviction strategy in use but fails in case of variable item sizes and costs. The greedy dual size variant considers the trade-off between costs, sizes and recency of use. The randomized version draws randomly from a smaller sample subset, chooses from and records only these samples. Hence, it achieves lower complexity since the item space being handled is reduced in size. Note that in this class of problems, however, the cache already knows what to fetch a-priori and the eviction decisions are the focus of attention.

Request routing, load balancing and content placement, while minimizing the linear communication costs, using distributed Nash Equilibrium and centralized integer programming approaches, is examined in [6, 7]. The former deals with placement of the entire contents of the cache at a single shot, while ensuring both local and global utility maximization, using a two step strategy. The latter examines the problem

of global bandwidth costs optimization, while developing co-operative caching techniques. However, these overlook the issue of online evictions decisions at the caches and finite capacity links.

In Chapter I, we have already addressed the resemblance that our CDN abstraction bears to the problem of scheduling in high-speed switches, and we briefly review work in this space. Tassiulas *et al.* initially proposed Max Weight Scheduling for Multi Hop Radio Networks with infinite buffers and interfering links in [8]. They proved that this policy is throughput-optimal and characterized the capacity region as the convex hull of all feasible schedules. This scheme commonly known as the Back-Pressure algorithm uses the queue length difference between the nodes as pressure of the links. Various extensions of this work that followed since are [9–13]. These papers proposed the use of the classic algorithm and explored the delays in the system for single down-link with variable connectivity, I.I.D on-off channels, multi-rate links and multi-hop wireless flows. Techniques of joint rate control, routing and scheduling using the same policy were developed in [14, 15]. Notice that "scheduling of a content type on a link" is similar to "scheduling of a link", "presence of an item at the cache" corresponds to the "existence of a link", and "scheduling a single content on a link in a time-slot" resembles the problem of "interference of links".

While none of these pieces of work directly applies to our problem, we will build upon the analytical techniques used in these papers as appropriate to our context.

CHAPTER III

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 $C_{sMax} = \max_s C_s$. 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.

A. Request Arrivals at Front-end 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 $\mathbb{P}(a_s^c[k] \leq A) > 1 - \delta_A \ \forall c, s, k$. In other words, the maximum probability of instantaneous arrivals for any request

queue overshooting this bound is given by δ. Finally, we define $\Lambda \triangleq \max_{c,s} \lambda_s^c$. The arrival processes must satisfy the following conditions that are necessary for stability:

$$\sum_{\forall c} \lambda_s^c < \sum_{\forall d} C_{sd}. \tag{3.1}$$

B. 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. 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 $\vec{p}[k]$. At each time instant, we allow each source to make a request for one type of content from each cache that it is connected to. We denote such a request by $\chi_{sd}^c[k] \in \{0, 1\}$, where it is understood that for each (s, d) pair at most one of the $\chi_{sd}^c[k]$ may be equal to 1. 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

$$\mu_{sd}^c[k] \triangleq \chi_{sd}^c[k] C_{sd}. \tag{3.2}$$

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 as

$$\tilde{\mu}_{sd}^c[k] \triangleq \min[\mu_{sd}^c[k], q_s^c[k]]. \tag{3.3}$$

Thus, the total number of departures that occur at s is $\tilde{\mu}_s^c[k] \triangleq \sum_{\forall d} \tilde{\mu}_{sd}^c[k]$. The evolution of the source queue containing requests for content c is then given by,

$$q_s^c[k+1] \;=\; q_s^c[k] + a_s^c[k] - \tilde{\mu}_s^c[k]. \tag{3.4}$$

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

$$\sum_{\forall c} \tilde{\mu}_s^c[k] \leq \sum_{\forall c} \mu_s^c[k] \leq C_s. \tag{3.5}$$

C. 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 $k = nD$

$$\chi_{sd}^c[k] \text{ can be chosen as 1 independent of } p_d^c[k]. \tag{3.6}$$

- At inter-refresh instants sources may only request pieces of content that are currently present in the caches, *i.e.*, for $k \neq nD$, $\forall \chi_{sd}^c[k] = 1$, we have,

$$\chi_{sd}^c[k] \times p_d^c[k] = 1. \tag{3.7}$$

We further define the request of an item from a cache, over all sources, as,

$$X_d^c[k] = \begin{cases} 1 & \text{if } \sum_{\forall s} \chi_{sd}^c[k] \geq 1 \\ 0 & \text{else.} \end{cases}$$

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.*

D. 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:

$$\chi_{sd}^c[k] + e_d^c[k] \leq 1. \tag{3.8}$$

- Only content that is present can be evicted, *i.e.* $\forall e_d^c[k] = 1$, we have,

$$e_d^c[k] \times p_d^c[k] = 1. \tag{3.9}$$

- The required number of evictions at a cache is given as,

$$\sum_c e_d^c[k] = \sum_c X_d^c[k] \left(1 - p_d^c[k]\right). \tag{3.10}$$

We summarize all the notations used in this work, in the following table,

Table I. Summary of notations.

Notation	**Definition**
E	Number of caches
S	Number of sources
B_d	Size of a cache d
D	Eviction Periodicity
N_d	Degree of cache d
N_s	Degree of source s
p_d^c	Presence of content c at cache d
l_{sd}	Physical link between s and d
C_{sd}	Capacity of link l_{sd}
q_s^c	Queue for content c at source s
e_d^c	Eviction of content c at cache d
χ_{sd}^c	Schedule of content c on link l_{sd}
a_s^c	Number of arrivals of content c at source s
λ_s^c	Arrival rate of content c at source s
η_s^c	Second Moment of arrivals of content c at source s
μ_{sd}^c	Scheduled number of departures of content c on link l_{sd}
$\tilde{\mu}_{sd}^c$	Actual number of departures of content c on link l_{sd}

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 (3.1), the expected values of all the request queues will remain finite. Such an objective implies that the delay suffered between request and service would also remain finite. Further, a quality of service metric that we consider is the expected value of the sum of all queues—the shorter the value, the smaller the expected delay.

CHAPTER IV

EXPLORING SCHEDULING ALGORITHMS
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 [10]. Unlike a switch, however, we have to take content placement, eviction and scheduling decisions refresh instants.

A. Formulating the Algorithm - A Heuristic Fluid Model

The intuition for a max weight scheduling policy arises, by examining the the heuristic fluid model of the system, using a Lyapunov function $V(t)$ involving the source queues lengths. We will attempt to use the Lyapunov stability criterion to determine the optimization expression, that may possibly allow for throughput-optimality in the system. The Lyapunov function we use is,

$$V(t) \;=\; \sum_{s,c} \frac{1}{2} \left(q_s^c(t)\right)^2 . \tag{4.1}$$

With some abuse of notation, we re-define certain discrete time variables for this section, to suit our needs as,

- $\tilde{\mu}_{sd}^c$ here is the fraction of time that the packets of content c departs on link l_{sd}.
- χ_{sd}^c is the fraction of time that the content c is scheduled on link l_{sd}.
- μ_{sd}^c is the fraction of time that C_{sd} packets of the content c are scheduled to be sent on the link l_{sd}.
- p_d^c is the fraction of time of content c is present at cache d.

- e_d^c is the fraction of time that the content c is scheduled to be evicted from cache d.

$$\begin{aligned}
\text{Departure rate} &= \sum_d \tilde{\mu}_{sd}^c \left(p_d^c \left(1 - e_d^c\right)\right) + \tilde{\mu}_{sd}^c \left(1 - p_d^c\right) \\
&= \sum_d \tilde{\mu}_{sd}^c \left(1 - p_d^c e_d^c\right). \\
\overset{(a)}{\Rightarrow} \text{Departure rate} &= \sum_d \tilde{\mu}_{sd}^c. \qquad (4.2) \\
\text{Hence giving } \dot{q}_s^c(t) &= \lambda_s^c - \sum_d \tilde{\mu}_{sd}^c, \qquad (4.3)
\end{aligned}$$

where (a) arises from (3.8). Simplifying the Lyapunov derivative,

$$\begin{aligned}
\dot{V}(t) &= \sum_{s,c} q_s^c(t) \dot{q}_s^c(t) \\
&= \sum_{s,c} q_s^c(t) \left(\lambda_s^c - \sum_d \tilde{\mu}_{sd}^c\right) \\
&= \sum_{s,c} q_s^c(t) \left(\lambda_s^c\right) - \sum_{s,c,d} q_s^c(t) \tilde{\mu}_{sd}^c.
\end{aligned}$$

The Lyapunov stability theorem for continuous time systems states that, the derivative of the Lyapunov function needs to be negative semi-definite, in order for the system to attain stable equilibrium. Hence to possibly achieve stable equilibrium in the system, the second term needs to be maximized.

Since $\tilde{\mu}_{sd}^c(t) \leq \mu_{sd}^c(t)$, by maximizing the scheduled service at every time instant, we maximize actual service at every time instant as well, differing only by a constant factor.

Returning to the discrete model, the schedule that the continuous time heuristic therefore suggests is given as,

$$\chi_{sd}^{c*}[k] = \arg\max_{\chi_{sd}^{c}} \sum_{s,c,d} q_s^c[k] C_{sd} \chi_{sd}^{c}[k]. \tag{4.4}$$

We refer to the above optimization as Max-Weight optimization *independent* of cache contents (MWI), which we use at the refresh instants. Note that the solution does not give information about the contents 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 incorporate 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 *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 formally 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,

$$\vec{\chi}^*[k] = \arg\max_{\chi} \mathcal{W}_{\mathcal{I}}(\vec{\chi}[k], \vec{\mathcal{Q}}[k]), \tag{4.5}$$

where

$$\mathcal{W}_{\mathcal{I}}(\vec{\chi}[k], \vec{\mathcal{Q}}[k]) \triangleq \sum_{\forall s,c,d} q_s^c[k] C_{sd} \chi_{sd}^c[k] \tag{4.6}$$

Fetch the requested content from the media vault and, if needed, evict contents arbitrarily from the cache such that the conditions (3.8) and (3.9) 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,

$$\vec{\hat{\chi}}[k] = \arg\max_{\chi} \mathcal{W}_{\mathcal{P}}(\vec{\chi}[k], \vec{\mathcal{Q}}[k], \vec{p}[k]), \tag{4.7}$$

where

$$\mathcal{W}_{\mathcal{P}}(\vec{\chi}[k], \vec{\mathcal{Q}}[k], \vec{p}[k]) \triangleq \sum_{\forall s,c,d} q_s^c[k] C_{sd} p_d^c[k] \chi_{sd}^c[k] \tag{4.8}$$

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

$$w^*[k] \triangleq \max_{\chi} \mathcal{W}_{\mathcal{I}}(\vec{\chi}[k], \vec{\mathcal{Q}}[k]). \tag{4.9}$$

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

$$\hat{w}[k] \triangleq \max_{\chi} \mathcal{W}_{\mathcal{P}}(\vec{\chi}[k], \vec{\mathcal{Q}}[k], \vec{p}[k]). \tag{4.10}$$

Following (3.2), we refer to the scheduled service associated with $\vec{\chi^*}[k]$ as $\vec{\mu^*}[k]$, and similarly that with $\vec{\hat{\chi}}[k]$ as $\vec{\hat{\mu}}[k]$.

B. 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*

$$\Delta V[k] = \mathbb{E}\left[V[k+1] - V[k] | \vec{\mathcal{Q}}[k]\right] \leq -\epsilon + bI_C, \tag{4.11}$$

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

$$\limsup_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} \mathbb{E}\left[f[i]\right] \leq \limsup_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} \mathbb{E}\left[g[i]\right]. \tag{4.12}$$

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

$$V[k] = \frac{1}{2} \sum_{\forall c,s} \left(q_s^c[k]\right)^2. \tag{4.13}$$

The drift of the Lyapunov function is given by

$$\begin{aligned}
\Delta V &= \mathbb{E}\left[V[k+1] - V[k] | \vec{\mathcal{Q}}[k]\right] \\
&= \frac{1}{2} \sum_{s,c} \mathbb{E}\left[\left(q_s^c[k+1]\right)^2 | \vec{\mathcal{Q}}[k+1]\right] - \mathbb{E}\left[\left(q_s^c[k]\right)^2 | \vec{\mathcal{Q}}[k]\right]. \qquad (4.14) \\
&\overset{(a)}{=} \frac{1}{2} \sum_{s,c} \mathbb{E}\left[\left(q_s^c[k] + a_s^c[k] - \tilde{\mu}_s^c[k]\right)^2 - \left(q_s^c[k]\right)^2 | \vec{\mathcal{Q}}[k]\right] \\
&= \frac{1}{2} \sum_{s,c} \mathbb{E}\left[2\left(q_s^c[k]\right)\left(a_s^c[k] - \tilde{\mu}_s^c[k]\right) | \vec{\mathcal{Q}}[k]\right] \\
&\qquad + \mathbb{E}\left[\left(a_s^c[k] - \tilde{\mu}_s^c[k]\right)^2 | \vec{\mathcal{Q}}[k]\right], \\
&= \sum_{c,s} \mathbb{E}\left[\left(q_s^c[k]\right)\left(\lambda_s^c - \tilde{\mu}_s^c\right) | \vec{\mathcal{Q}}[k]\right] - \sum_{c,s} \mathbb{E}\left[a_s^c[k]\tilde{\mu}_s^c[k] | \vec{\mathcal{Q}}[k]\right] \\
&\qquad + \frac{1}{2} \sum_{c,s} \mathbb{E}\left[a_s^{c2}[k] | \vec{\mathcal{Q}}[k]\right] + \frac{1}{2} \sum_{c,s} \mathbb{E}\left[\tilde{\mu}_s^{c2}[k] | \vec{\mathcal{Q}}[k]\right] \\
&= \sum_{c,s} \mathbb{E}\left[\left(q_s^c[k]\right)\left(\lambda_s^c - \tilde{\mu}_s^c\right) | \vec{\mathcal{Q}}[k]\right] + B_1, \qquad (4.15)
\end{aligned}$$

where (a) arises using (3.4). B_1 is a finite, bounded, positive quantity defined as,

$$B_1 = \frac{1}{2}\sum_{c,s}\mathbb{E}\left[a_s^{c2}[k]|\vec{\mathcal{Q}}[k]\right] + \frac{1}{2}\sum_{c,s}\mathbb{E}\left[\tilde{\mu}_s^{c2}|\vec{\mathcal{Q}}[k]\right] - \sum_{c,s}\mathbb{E}\left[\lambda_s^c\tilde{\mu}_s^c[k]|\vec{\mathcal{Q}}[k]\right], \tag{4.16}$$

We employ the following simplification in our drift,

$$\begin{aligned}
&\sum_{c,s}\mathbb{E}\left[q_s^c[k]\left(\mu_s^c[k]-\tilde{\mu}_s^c[k]\right)|\vec{\mathcal{Q}}[k]\right]\\
=\;&\sum_{c,s}\mathbb{E}\left[\tilde{\mu}_s^c[k]\left(\mu_s^c[k]-\tilde{\mu}_s^c[k]\right)|\vec{\mathcal{Q}}[k]\right]\\
=\;&\sum_{c,s}\mathbb{E}\left[\tilde{\mu}_s^c[k]\mu_s^c[k]|\vec{\mathcal{Q}}[k]\right]-\mathbb{E}\left[\tilde{\mu}_s^{c2}[k]|\vec{\mathcal{Q}}[k]\right]\\
\leq\;&\sum_{c,s}\mathbb{E}\left[\min\left[\mu_s^{c2}[k],C_s\tilde{\mu}_s^c[k]\right]|\vec{\mathcal{Q}}[k]\right]-\mathbb{E}\left[\tilde{\mu}_s^{c2}[k]|\vec{\mathcal{Q}}[k]\right]\\
\overset{(a)}{\leq}\;&\sum_{c,s}\min\left[\mathbb{E}\left[\mu_s^{c2}[k]|\vec{\mathcal{Q}}[k]\right],\mathbb{E}\left[C_s\tilde{\mu}_s^c[k]\right]|\vec{\mathcal{Q}}[k]\right]-\sum_{c,s}\mathbb{E}\left[\tilde{\mu}_s^{c2}[k]|\vec{\mathcal{Q}}[k]\right]\\
\overset{(b)}{\leq}\;&\min\left[\sum_{c,s}\mathbb{E}\left[\mu_s^{c2}[k]|\vec{\mathcal{Q}}[k]\right],\sum_{cs}\mathbb{E}\left[C_s\tilde{\mu}_s^c[k]|\vec{\mathcal{Q}}[k]\right]\right]\\
&\qquad-\sum_{c,s}\mathbb{E}\left[\tilde{\mu}_s^{c2}[k]|\vec{\mathcal{Q}}[k]\right]\\
\overset{(c)}{\leq}\;&\min\left[\sum_{cs}C_s^2,\sum_{s}\mathbb{E}\left[C_s\tilde{\mu}_s[k]|\vec{\mathcal{Q}}[k]\right]\right]-\sum_{c,s}\mathbb{E}\left[\tilde{\mu}_s^{c2}[k]|\vec{\mathcal{Q}}[k]\right].
\end{aligned}\tag{4.17}$$

(a) is obtained from Jensen's inequality and concavity of the min[.,.] operator, (b) from the min function property that the sum of the min is less than or equal to the min of the sum and (c) arises from (3.5). Hence using the result (4.17) in (4.15)

yields,

$$\begin{aligned}
\Delta V &= \sum_{c,s} \mathbb{E}\left[\left(q_s^c[k]\right)\left(\lambda_s^c - \tilde{\mu}_s^c\right)|\vec{\mathcal{Q}}[k]\right] + B_1 \\
&\leq \sum_{c,s} \mathbb{E}\left[q_s^c[k]\left(\lambda_s^c - \mu_s^c[k]\right)|\vec{\mathcal{Q}}[k]\right] - \sum_{c,s} \mathbb{E}\left[\tilde{\mu}_s^{c2}[k]|\vec{q}[k]\right] \\
&\qquad + B_1 + \min\left[\sum_{c,s} C_s^2, \sum_s \mathbb{E}\left[C_s\tilde{\mu}_s[k]|\vec{\mathcal{Q}}[k]\right]\right] \\
&\leq B_1 + B_2 + \sum_{c,s} \mathbb{E}\left[q_s^c[k]\left(\lambda_s^c - \mu_s^c[k]\right)|\vec{\mathcal{Q}}[k]\right],
\end{aligned}$$

where B_2 is a finite, bounded, positive quantity defined as,

$$\begin{aligned}
B_2 &= \min\left[\sum_{c,s} C_s^2, \sum_s \mathbb{E}\left[C_s\tilde{\mu}_s[k]|\vec{\mathcal{Q}}[k]\right]\right] \\
&\qquad - \sum_{c,s} \mathbb{E}\left[\tilde{\mu}_s^{c2}[k]|\vec{\mathcal{Q}}[k]\right].
\end{aligned} \tag{4.18}$$

Hence, we derive the drift as,

$$\Delta V[k] \leq B_1 + B_2 + \sum_{c,s} \mathbb{E}\left[\left(q_s^c[k]\right)\left(\lambda_s^c - \mu_s^c[k]\right)|\vec{\mathcal{Q}}[k]\right], \tag{4.19}$$

where B_1 and B_2 are defined in (4.16) and (4.18).

We use a similar line of reasoning that was used earlier in [10] and [9] 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 (3.1). 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 $\vec{\lambda} + \vec{\epsilon} \leq \sum_i \beta_i\vec{\mu}_i$. The scheduled service at time k also satisfies $\mu^*[k] \leq \sum_i \beta_i\vec{\mu}_i$. We then deduce from (3.2) and (4.5) that,

$$\sum_{\forall s,c} \mathbb{E}\left[q_s^c\left(\lambda_s^c - \mu_s^{c*}[k]\right)|\vec{\mathcal{Q}}[k]\right] \leq -\epsilon \sum_{\forall s,c} q_s^c[k]. \tag{4.20}$$

Examining the drift in (4.19) when $\mu_s^c[k] = \mu_s^{c*}[k]$,

$$\begin{aligned}\Delta V &\leq B_1 + B_2 \\ &\quad + \sum_{c,s} \mathbb{E}\left[\left(q_s^c[k]\right)\left(\lambda_s^c - \mu_s^{c*}\right)|\vec{\mathcal{Q}}[k]\right] \\ &\overset{(a)}{\leq} B_1 + B_2 - \epsilon \sum_{c,s} q_s^c[k], \end{aligned} \tag{4.21}$$

where (a) follows from (4.20). Hence, except for a finite subset of $\mathbf{Q}$, 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 (4.13) 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 (4.19) with $\mu_s^c[k] = \hat{\mu}_s^c[k]$ yielding,

$$\Delta V \leq B_1 + B_2 + \sum_{c,s} \mathbb{E}\left[\left(q_s^c[k]\right)\left(\lambda_s^c - \hat{\mu}_s^c[k]\right)|\vec{\mathcal{Q}}[k]\right]. \tag{4.22}$$

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,

$$\begin{aligned} q_s^c[k] &\geq q_s^c[k-1] - C_{sMax} \\ \Rightarrow q_s^c[k] &\geq q_s^c[lD] - C_{sMax}(k - lD + 1). \end{aligned} \tag{4.23}$$

- Since the maximum number of arrivals to the system is upper bounded by $A \geq 0$ with probability $(1 - \delta_A)$ at any time instant,

$$\begin{aligned} q_s^c[k-1] &\geq q_s^c[k] - A \\ \Rightarrow q_s^c[lD] &\geq q_s^c[k] - A(k - lD). \end{aligned} \tag{4.24}$$

We then have with probability $(1 - \delta_A)$,

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

where (a) follows from (4.23), and (b) from (4.24). Thus, we have just shown that with probability $(1 - \delta_A)$,

$$-\sum_{c,s} \hat{\mu}_s^c[k] q_s^c[k] \leq -\sum_{c,s} \mu_s^{c*}[k] q_s^c[k] + f(D), \tag{4.25}$$

where

$$f(D) = C_{tot} C_{sMax}(k - lD + 1) + C_{tot} A(k - lD). \tag{4.26}$$

Now, from (4.22) and (4.25) we obtain,

$$\begin{aligned}
\Delta V &\leq B_1 + B_2 + (1-\delta_A) f(D) \\
&\quad + (1-\delta_A) \sum_{c,s} \mathbb{E}\left[(q_s^c[k]) \, (\lambda_s^c - \mu_s^{c*}[k]) \, | \vec{\mathcal{Q}}[k] \right] \\
&\quad + \delta_A \sum_{c,s} \mathbb{E}\left[(q_s^c[k]) \, (\lambda_s^c - \hat{\mu}_s^c[k]) \, | \vec{\mathcal{Q}}[k] \right] \\
&\overset{(c)}{\leq} B_1 + B_2 + (1-\delta_A) f(D) \\
&\quad + (\delta_A \Lambda - (1-\delta_A)\epsilon) \sum_{c,s} q_s^c[k] \\
&\leq B_1 + B_2 + (1-\delta_A) f(D) \\
&\quad - \alpha_A \sum_{c,s} q_s^c[k], \qquad (4.27)
\end{aligned}$$

where

$$\alpha_A \triangleq (-\delta_A \Lambda + (1-\delta_A)\epsilon) \, . \qquad (4.28)$$

(c) arises from (4.20) and from the fact that,

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

By selecting A in such a way that $\alpha_A > 0$, the drift is negative except for a finite subset of $\mathbf{Q}$. Therefore, combining the results of Theorem 2 and the relation (4.27), 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 characterizes the delays, and we now find upper bounds on the expected queue lengths.

Corollary 1. *The sum of the queue backlogs in the system using the PMW Algorithm with unit refresh periodicity satisfies*

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

Proof. We use the steady state condition of the Markov chain given in (4.12) to find the expected queue lengths. From Theorem 1 and (4.12) we have,

$$0 \leq B_1 + B_2 - \epsilon \sum_{c,s} \mathbb{E}\left[q_s^c[k]\right]. \tag{4.29}$$

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 (4.18) and (4.16). 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. A similar result has been obtained in [11] in the context of wireless networks where, for multi-rate systems, it has been proved that the source queue lengths obtained through any scheduling policy increases with the system capacities and the network size.

Corollary 2. *The sum of the queue lengths for the CDN that uses the PMW Algorithm with period D satisfies*

$$\begin{aligned}\sum_{c,s} \mathbb{E}\left[q_s^c[k]\right] \;\leq\; & \frac{\sum_{s,c} \eta_s^c}{2\alpha_A} - \frac{3\sum_{s,c} \lambda_s^{c2}}{2\alpha_A} \\ & + \frac{\min\left[\sum_{c,s} C_s^2, \sum_s C_s \lambda_s\right]}{\alpha_A} \\ & + \frac{(1-\delta_A)\, f(D)}{\alpha_A},\end{aligned}$$

where $f(D)$ is given by (4.26).

Proof. The proof follows from (4.27) in the same manner as Corollary 1. □

From (4.26) 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 2, 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.

Discussion

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.

CHAPTER V

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.

A. Min-weight Eviction Policy

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,

$$\begin{aligned} \sum_{c,s,d} C_{sd} q_s^c[k] \chi_{sd}^{cM}[k] &\geq \sum_{c,s,d} C_{sd} q_s^c[k] \chi_{sd}^{cR}[k] \\ \Rightarrow \sum_{c,s} q_s^c[k] \mu_s^{cM}[k] &\geq \sum_{c,s} q_s^c[k] \mu_s^{cR}[k] \qquad (5.1) \\ \Rightarrow \sum_{c,s} q_s^c[k] \left(\lambda_s^c - \mu_s^{cM}[k]\right) &\leq \sum_{c,s} q_s^c[k] \left(\lambda_s^c - \mu_s^{cR}[k]\right). \qquad (5.2) \end{aligned}$$

Hence, using (5.2) and from the expression for the drift in (4.19), we deduce that except for a finite subset of $\mathbf{Q}$,

$$\Delta V^M \leq \Delta V^R \leq 0, \qquad (5.3)$$

and it follows from an argument similar to Corollaries 1 and 2 that the queue lengths under M are shorter than for R. Thus, a greater weight of the schedule at each time instant results 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:

$$p_d^c[k] = (1 - e_d^c[lD])\,(p_d^c[lD]) + (1 - p_d^c[lD])\,(X_d^{c*}[lD])\,, \tag{5.4}$$

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 (5.1) 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 (4.7) and since the second term in (5.4) is the same for both policies,

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

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 (3.8). 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,

$$e_d^{c*}[k] = \arg\min_{e_d^{c*}} \sum_{c,s} \left(q_s^c[k] p_d^c[k] C_{sd}\right) e_d^c[k], \tag{5.5}$$

$\forall d, k$, subject to condition (3.8).

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 4. *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. □

B. Joint Scheduling and Eviction Policy

The PMW policy with Min-Weight evictions is a two-step approach, which first handles scheduling at the nodes, and then the evictions at the caches. We would like to explore the performance of a policy that provides a one-shot, joint solution to the scheduling-eviction problem. Since we have just defined the PMW policy with Min-Weight Evictions, we use the same nature of scheduling and eviction decisions made in that policy. We then formulate a joint optimization problem that renders the coupled solution of what content needs to be scheduled at the links, and what

content needs to be evicted from the caches. Such a scheduling-eviction optimization (SE) can be written as,

$$\max_{\chi^c_{sd}, e^c_d} \sum_{csd} \left(EC_{sd} q^c_s[k]\right) \chi^c_{sd}[k] - \sum_{cd} \left(\sum_{s} \left(q^c_s[k] C_{sd} p^c_d[k]\right) \right) e^c_d[k]. \tag{5.6}$$

We now propose a Joint Scheduling and Eviction policy (JSE), that incorporates the SE solution at the refresh instants, and MWP at the inter-refresh instants.

Algorithm 3: Joint Scheduling and Eviction

At the refresh instants, for all $k = nD$, schedule and evict content from the cache, based on SE, subject to (3.8),

$$(\chi^{c*}_{sd}[k], e^{c*}_d[k]) = \arg \max_{\chi^c_{sd}, e^c_d} \sum_{csd} (EC_{sd} q^c_s[k]) \chi^c_{sd}[k] - \sum_{cd} \left(\sum_{s} (q^c_s[k] p^c_d[k] C_{sd}) \right) e^c_d[k]. \tag{5.7}$$

Fetch the missing contents as needed.

At the inter-refresh instants, for all $k \neq nD$, schedule based on MWP as in Algorithm 1.

Since we have no provable results on JSE, we will study the performance of this heuristic algorithm through our simulations in Chapter VII. We will then attempt to gain an insight into the usefulness of a joint approach over a two-step one, if it exists.

CHAPTER VI

EXPLORING SCHEDULING ALGORITHMS

THE ITERATIVE PERIODIC MAX WEIGHT ALGORITHM

An important observation on LQF scheduling in the context of wireless networks in [3] is that it could result in capacity wastage if a queue scheduled for service does not have enough packets to utilize the entire capacity. This loss intensifies with higher system capacities. The paper then proposes an iterative solution – Iterative Longest Queue First policy (ILQF), to lessen such wastage.

This problem applies to our context as well. We therefore attempt a similar iterative scheme to ensure non-zero service on all the links, at all instants. We formulate this scheme in such a way that it further guarantees the same amount of service as provided by the PMW scheduling policy, had it been in use. We now propose the IPMW - Iterative Periodic Max-Weight Scheduling policy which uses the Iterative variants of MWI and MWP (IMWI / IMWP).

We introduce some additional terminology for our new policy. Let us refer to the schedules and scheduled service 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.* $C_{sd} \leq C_{s(d-1)}\ \forall d \in \mathcal{E}^s$. $\mathcal{E}^s_{PERM}$ is set of all permuted cache orderings and $\mathcal{E}^s_j \in \mathcal{E}^s_{PERM}\ \forall 1 \leq j \leq E!$. $\mathcal{F}^s[k]$ is the set of all caches that serviced the selected queue until the current iteration. The algorithm is described as follows.

Algorithm 4: 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, **do**

$$\begin{aligned} q_s^{cI*}[k] &= \left(q_s^{cI*}[k] - C_{sd}\right)^+, \qquad (6.1) \\ \chi_{sd}^{cI*}[k] &= 1, \\ \mathcal{F}^s[k] &= \mathcal{F}^s[k] \cup \{d\}. \end{aligned}$$

Fetch the missing content and evict arbitrarily subject to (3.8) and (3.9).

if $q_s^{cI*}[k] = 0$, **then**

break.

end if

end for

until $\mathcal{F}^s[k] = \mathcal{E}^s$.

Algorithm 4 continued.

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**

$$\begin{aligned} \mu_{sj}^e[k] &= \mu_{sj}^e[k] + \min[C_{sd}, q_{sj}^c[k]], \\ q_{sj}^c[k] &= \left(q_{sj}^c[k] - C_{sd}\right)^+, \\ \chi_{sdj}^{ce}[k] &= 1, \\ \mathcal{F}_j^{se}[k] &= \mathcal{F}_j^{se}[k] \cup \{d\}. \end{aligned} \tag{6.2}$$

end if

end for

end for

end for

Algorithm 4 continued.

for each $\mathcal{E}_j^s \in \mathcal{E}_{PERM}^s$, **do**

We now continue with scheduling the remaining sets of unserviced links - $\mathcal{E}_j^s \backslash \mathcal{F}_j^{se}[k]$,

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 (3.7) for that cache d. Estimate the throughput, update the queue and schedule the link as,

$$\begin{aligned} \mu_{sj}^e[k] &= \mu_{sj}^e[k] + \min[C_{sd}, q_{sdj}^{ce}[k]], \\ q_{sdj}^{ce}[k] &= \left(q_{sdj}^{ce}[k] - C_{sd}\right)^+, \\ \chi_{sdj}^{ce}[k] &= 1, \\ \mathcal{F}_j^{se}[k] &= \mathcal{F}_j^{se}[k] \cup \{d\}. \end{aligned} \tag{6.3}$$

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

A. Stability and Performance Analysis of the IPMW Policy

Theorem 5. *The Iterative Periodic Max-Weight scheduling policy is throughput optimal, for all finite refresh periodicities. The average queue length is no greater than that of the Periodic Max-Weight scheduling policy.*

Proof. From the definition of IMWI, it is noticeable that,

$$\tilde{\mu}_s^{cI*}[nD] \;\geq\; \tilde{\mu}_s^{c*}[nD] \;\forall c, s, n. \tag{6.4}$$

Further from the definition of IMWP, $\forall\ \hat{\chi}_{sd}^{c}[k] = 1$,

$$\hat{\tilde{\mu}}_s^{cI}[k] \;=\; \hat{\tilde{\mu}}_s^{c}[k], \tag{6.5}$$

$$\Rightarrow \hat{\tilde{\mu}}_s^{cI}[k] \;\geq\; \hat{\tilde{\mu}}_s^{c}[k]. \tag{6.6}$$

Using (6.6) and (6.4), it is evident that the real throughput on the links for the IPMW policy, is greater or, at worst the same as the throughput for the PMW policy at all instants. Hence for all k, c, s,

$$\tilde{\mu}_s^{cI}[k] \;\geq\; \tilde{\mu}_s^{c}[k]. \tag{6.7}$$

Re-writing the drift in 4.19 to involve the real service instead of scheduled service for the IPMW policy,

$$\begin{aligned}
\Delta V &= B_1 + \sum_{c,s} \mathbb{E}\left[\left(q_s^c[k]\right)\left(\lambda_s^c - \tilde{\mu}_s^c[k]\right)|\vec{\mathcal{Q}}[k]\right] &(6.8)\\
&= B_1 + \sum_{c,s} \mathbb{E}\left[\left(q_s^c[k]\right)\left(\lambda_s^c - \tilde{\mu}_s^{cI}[k]\right)|\vec{\mathcal{Q}}[k]\right]\\
&\leq B_1 + \sum_{c,s} \mathbb{E}\left[\left(q_s^c[k]\right)\left(\lambda_s^c - \tilde{\mu}_s^c[k]\right)|\vec{\mathcal{Q}}[k]\right].\\
&\Rightarrow \Delta V^I \leq \Delta V^* \leq 0. &(6.9)
\end{aligned}$$

Hence the proof follows directly from that of Theorem 3. □

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 5: IPMW Scheduling with Min-Weight Evictions

At the refresh instants, for all $k = lD$, schedule based on IMWI as in Algorithm 4. 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 (3.8).

At the inter-refresh instants, for all $k \neq lD$, schedule based on IMWP as in Algorithm 4.

B. A Heuristic Variant of IPMW

Perfect Iterative Periodic Max Weight Algorithm

The greater drifts obtained using the iterative PMW algorithm indicate that iterative queue updates, and ensuring non-zero service on all the links, can rid us of the capacity wastage problem. Now we propose a heuristic variation of IPMW, wherein we ignore the question of throughput-optimality of the policy, and try to achieve maximum possible service on each of the links, at all times. We formulate the Perfect Iterative Periodic Max Weight Algorithm, which incorporates the Perfect Iterative variants of MWI and MWP (PIMWI / PIMWP), at the refresh and inter-refresh instants respectively.

Let us refer to the schedules and the scheduled service as $\chi_{sd}^{cPI*}[k]$ / $\mu_{sd}^{cPI*}[k]$ for $k = lD$ and $\hat{\chi}_{sd}^{cPI}[k]$ / $\hat{\mu}_{sd}^{cPI}[k]$ for $k \neq lD$.

Algorithm 6: Perfect Iterative Periodic Max-Weight Scheduling

At the refresh instants, for all $k = lD$, at each source s,

Find the longest queue $q_s^{cPI*}[k]$.

for each $d \in \mathcal{E}^s$ (in order), update the queue and schedule the link as, **do**

$$\begin{aligned} q_s^{cPI*}[k] &= \left(q_s^{cPI*}[k] - C_{sd}\right)^+, \\ \chi_{sd}^{cPI*}[k] &= 1. \end{aligned} \tag{6.10}$$

Fetch the missing content and evict arbitrarily subject to (3.8) and (3.9).

end for

At the inter-refresh instants, for all $k \neq lD$, at each source s, we initially estimate schedules, queue updates, throughputs without actual implementation,

for each $\mathcal{E}_j^s \in \mathcal{E}_{PERM}^s$, **do**

for each $d \in \mathcal{E}_j^s$ (in order), **do**

Find the longest queue $q_{sdj}^{ce}[k]$ subject to (3.7) for that cache d. Estimate the throughput $\mu_{sj}^e[k]$, update the queue and schedule the link as,

$$\begin{aligned} \mu_{sj}^e[k] &= \mu_{sj}^e[k] + \min[C_{sd}, q_{sdj}^{ce}[k]], \\ q_{sdj}^{ce}[k] &= \left(q_{sdj}^{ce}[k] - C_{sd}\right)^+, \\ \chi_{sdj}^{ce}[k] &= 1. \end{aligned} \tag{6.11}$$

end for

Compute the net estimated throughput for this cache ordering as $\mu_j^e[k] = \sum_s \mu_{sj}^e[k]$.

end for

Algorithm 6 continued.

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

We will examine the performance of this policy later via simulations in Chapter VII.

CHAPTER VII

SIMULATIONS

We now illustrate the insights from our analytical model by simulating our CDN abstraction using C++. We developed a network model with defined attributes such as the network capacities, arrival rates, eviction periodicities and cache sizes. It is composed of a clocking mechanism and event coordinator that synchronizes request arrivals, scheduling, request routing, service, content fetches and evictions and queue updates at the frontend and rearend nodes. We use the Open Source Linear Programming Solver - lp_solve (Version: 5.5.0.15) to compute the SE schedules and evictions which is a binary integer program.

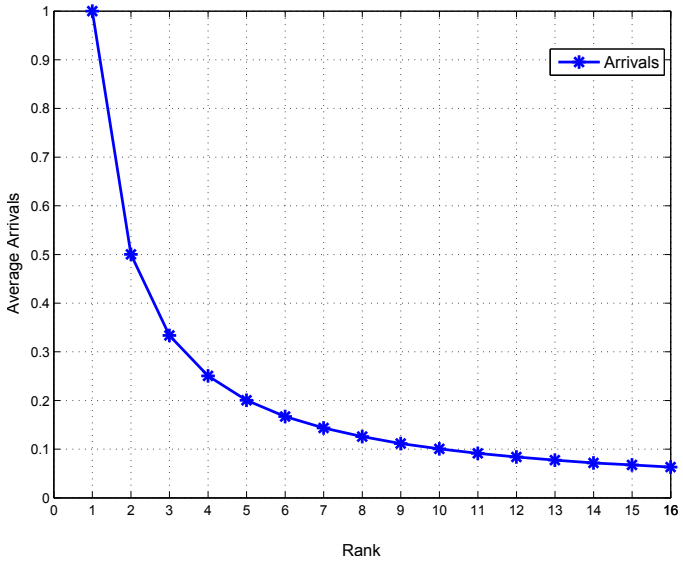


Fig. 2. Zipf arrival process at a front-end node showing the normalized arrivals of various content types versus their popularity ranks.

The popularity of each piece of content follows a Zipf distribution, with some

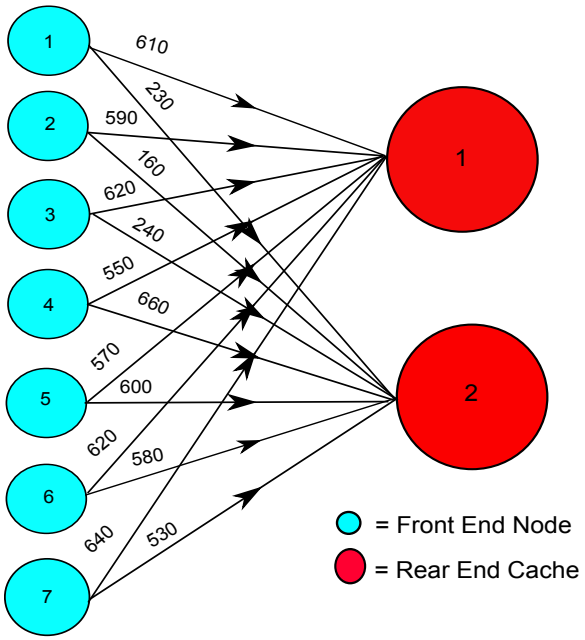


Fig. 3. A fully connected network topology with all the front-end nodes capable of routing requests to any of the rear-end nodes.

dominant ones and a large number of less popular ones [16, 17]. Similarly, the popularity of content can vary by geographical region, thus creating different distributions at each of the frontend nodes. In our simulation, each source node with fixed arrival rates has a different rank distribution over content, and the arrivals take place proportional to these ranks. A typical rank distribution at a node is illustrated in Figure (2), with the average normalized arrivals for the different content types in the increasing order of ranks.

We randomly assign the initial cache contents, network capacities and arrival rates for the network.

We are interested in the following scenarios:

1. A fully connected CDN, with $S = 7, E = 2, C = 16$ and $B = 10$ in Figure 3, with link capacities chosen arbitrarily. We derived our analytical results for this

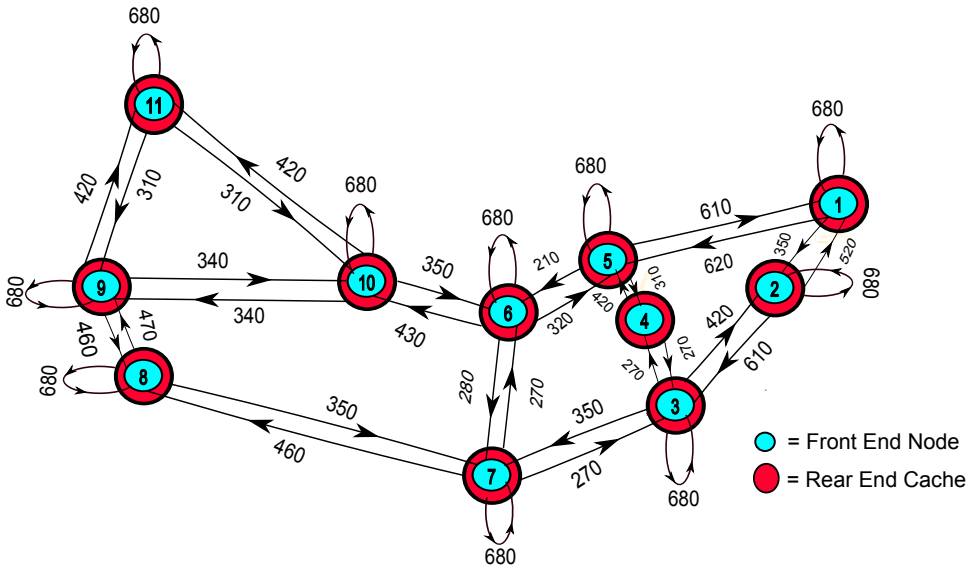
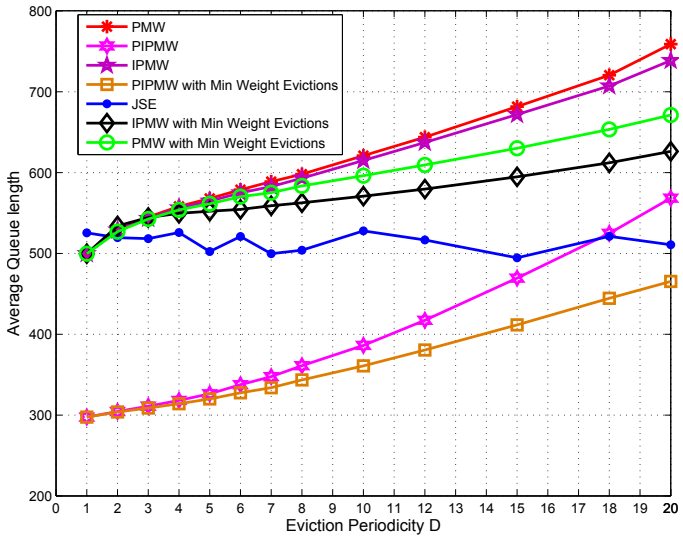


Fig. 4. The Abilene network topology where each node represents a source and an associated cache. Sources may route requests to neighboring caches.

topology, although they are easily generalized to arbitrary topologies.

2. A CDN topology that follows the Abilene network [18], illustrated in Figure 4. Sources can access a local cache, as well as other neighboring caches as single-hop flows only. Links are capacitated, and we have $S = 11, E = 11, C = 16$ and $B = 5$. We expect all our analytical insights to also apply in this case. We varied the eviction periodicity for these two scenarios as $1 \leq D \leq 20$ time units, with the total time of simulation $T = 12000$ time units.

3. The Abilene network in which refresh periodicities are different at different caches. We set the baseline refresh periods arbitrarily 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.



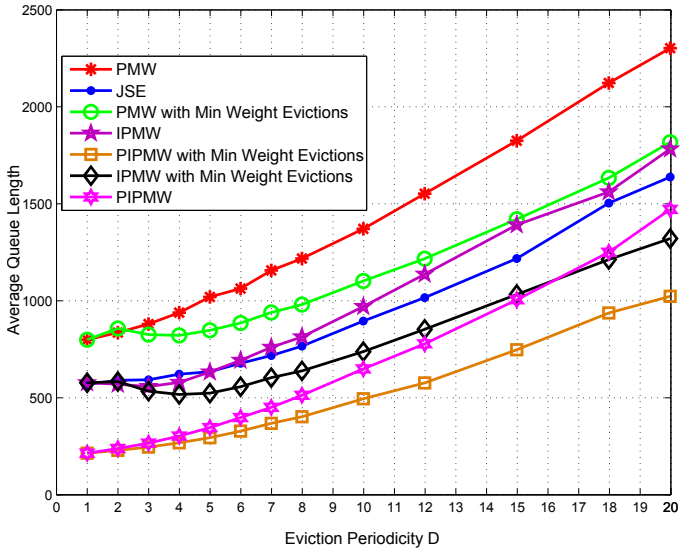
(a) Fully Connected Topology

Fig. 5. Variation of average queue length with refresh periodicity for different algorithms.

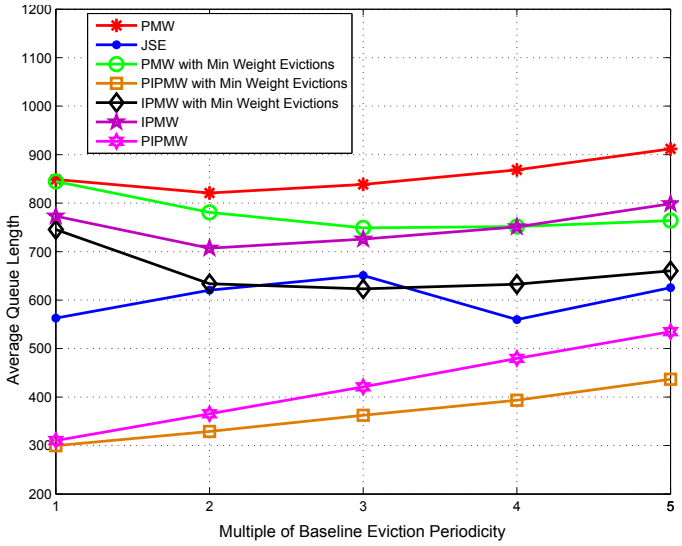
A. Variation of Delays with Eviction Periodicity

We conduct a first set of simulations to illustrate the performance of our routing-placement-eviction policies as the refresh periodicity increases. We had tied large (negative) Lyapunov drifts with shorter queue lengths. We now show that such design is indeed valid from a performance standpoint. In Corollary 2, we had discussed the increase of the queue backlogs in the system with the refresh periodicity, due to the factor $f(D)$. Further in Theorem 5, we state the IPMW policy results in the same or larger drifts (more departures) than PMW, indicative of smaller delays. In Chapter V, as per our discussion, Min-Weight eviction strategies with any scheduling algorithm will always engender shorter queues. We would also like to examine the performance of JSE and PIPMW in their own regimes.

In Figures 5 and 6, we see that for any network, all scheduling algorithms expe-

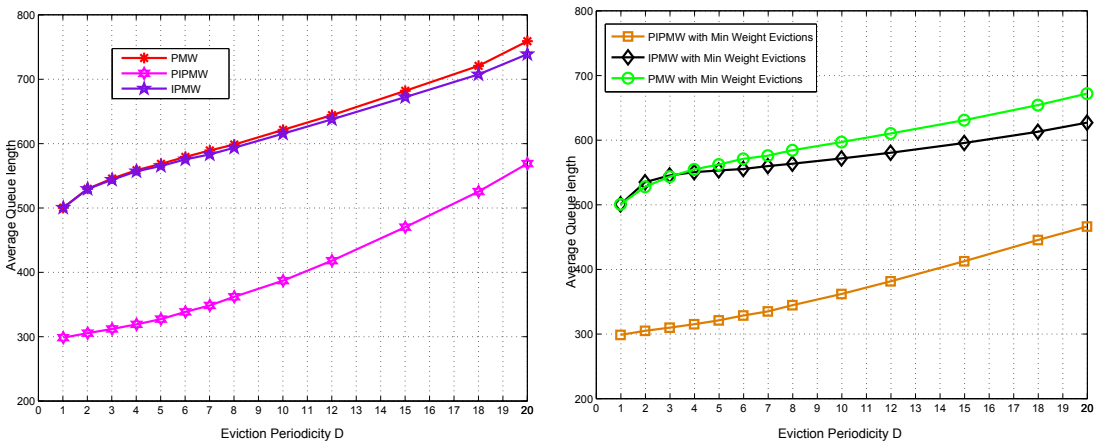


(a) Abilene Network Topology



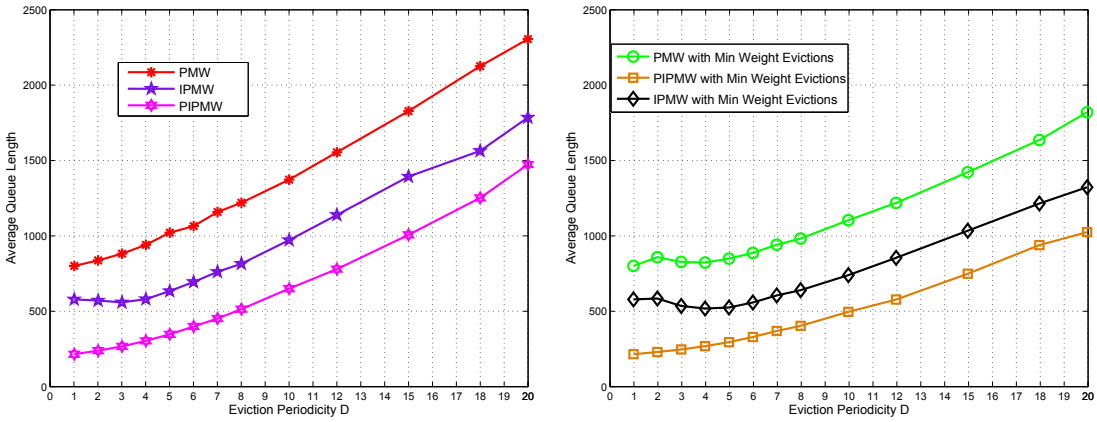
(b) Heterogeneous Eviction Periodicities

Fig. 6. Variation of average queue length with refresh periodicity for different algorithms.



(a) Random Eviction Strategy (b) Min-Weight Eviction Strategy

Fig. 7. Performance comparison for eviction strategies for the fully connected topology.



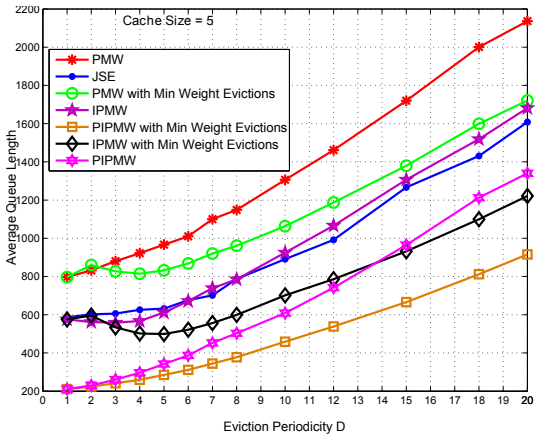
(a) Random Eviction Strategy (b) Min-Weight Eviction Strategy

Fig. 8. Performance comparison for eviction strategies for the Abilene topology.

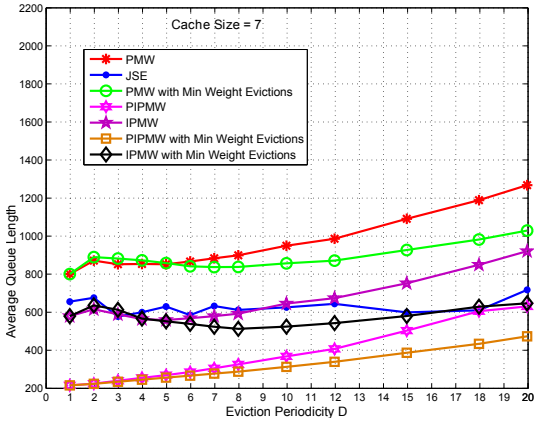
rience higher delays with larger refresh periodicities. To get a better picture, Figures 7 and 8 give a clear comparison of all iterative and non-iterative algorithms for the two eviction strategies. The PMW policy coupled with Random Evictions has the largest queue lengths, which corresponds to the 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 (IPMW) outperform the non-iterative ones; again the result follows from Lyapunov drift arguments. Surprisingly, the heuristic JSE policy that implements a joint version of the PMW with Min-Weight evictions, gives lower delays in the system than the two-step approach. This give us us new insight into the potential advantage that a coupled approach could have over the two-step approach. Finally, the heuristic PIPMW algorithm, that ensures full service on all links, does better than IPMW, independent of the eviction strategy employed.

B. Variation of Delays with Cache Size

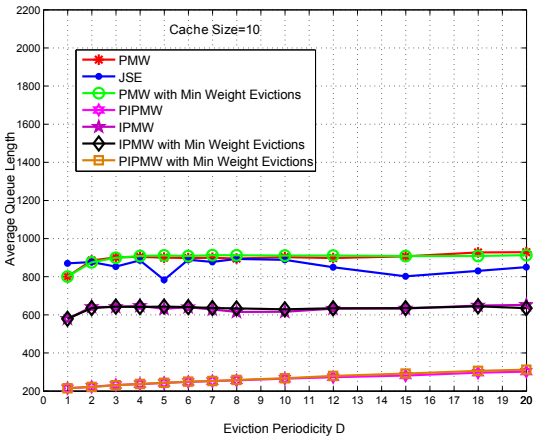
We now explore performance variation with the cache size. Intuition suggests that the decision to cache the "useful" objects and the eviction technique employed play an important role only when $B > N_d$,. We studied this facet of the problem using the Abilene network with homogeneous eviction periods, and varied the cache size as $B \in \{5, 7, 10\}$ along with D.



(a) Cache Size $B = 5$



(b) Cache Size $B = 7$



(c) Cache Size $B = 10$

Fig. 9. Variation of average queue lengths with eviction periodicity for variable cache sizes.

In Figure 9(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.

CHAPTER VIII

CONCLUSION

In this work 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). Further using insights obtained, we then began our search for better algorithms that would yield short queues as well, and hence delays. 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 initially developed a scheduling algorithm that used instantaneous queue length information and performed random evictions. For the same scheduling algorithm, we then developed an eviction technique that used state information. We illustrated its superior performance over the former. We also created a regime of algorithms that uses iteratively updated queue lengths for scheduling, employing the same eviction strategies developed earlier. We showed that these are more efficient than their non-iterative counterparts, yielding lower delays. We discussed the use of two heuristic algorithms in each regime, which could potentially exhibit desirable performance. Finally, we validated all our results through simulations.

Our current work only accounted for requests with a soft delay tolerance. Future extensions of this work could possibly include streaming traffic with requests that have hard delay constraints, and which are dropped if such a constraint cannot be met. Further, we observed through our simulations that, a coupled approach of scheduling and evictions is somehow advantageous, giving rise to relatively shorter delays than the non-iterative two-step algorithms. The inherent conservative nature

of the heuristic scheme could be the causative factor. Design of joint, low complexity, throughput-optimal routing and eviction algorithms is yet another area of interest to be explored.

REFERENCES

[1] C. Labovitz, D. McPherson, S. Iekel-Johnson, F. J. J.Oberheide, and M. Karir, "Atlas Internet Observatory 2009 annual report," in *Proc. NANOG-47*, Dearborn, Michigan, Jun. 2009, pp. 1–32.

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

[3] S. Bodas, S. Shakkottai, L. Ying, and R. Srikant, "Scheduling in multi-channel wireless networks: Rate function optimality in the small-buffer regime," in *Proc. ACM SIGMETRICS'09*, Seattle, WA, Jun. 2009, pp. 121–132.

[4] 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, pp. 193–206.

[5] 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.

[6] 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.

[7] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. IEEE INFOCOM 2010*, San Diego, CA, Mar. 2010, pp. 1–9.

[8] 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.

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

[10] L. Tassiulas and A. Ephremides, "Dynamic server allocation to parallel queues with randomly varying connectivity," *IEEE Transactions on Information Theory*, vol. 39, no. 2, pp. 466–478, Mar. 1993.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] X. Lin and N. Shroff, "Joint rate control and scheduling in multihop wireless

networks," presented at the 43rd IEEE Conference on Decision and Control (CDC 2004), Paradise Islands, Bahamas, Dec. 2004.

[16] 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, pp. 126–134.

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

[18] Internet2, accessed on 18 June 2010. [Online]. Available: http://www.internet2.edu/

VITA

Meghana Mukund Amble received the degree of Bachelor of Technology in electrical and electronics engineering from the National Institute of Technology, Karnataka, India in May 2007. She then received the Master of Science degree in electrical engineering from Texas A&M University in December 2010. Her professional experience includes a year (July 2007 - June 2008) as a Firmware Engineer in the Systems and Technology Group at IBM, India Private Limited (Bangalore, India) and, as a summer intern (June 2009 - August 2009) at Cisco Systems, Inc. (San Jose, USA). Her research interests lie in control theory, computer networks and development of protocols and algorithms for the transport layer and above.

Ms.Amble may be reached at:

c/o Dr. Srinivas Shakkottai

Computer Engineering Group,

Department of Electrical & Computer Engineering,

Texas A&M University,

College Station, TX 77843-3259

Email : meghana.amble@gmail.com