Orchestrating Massively Distributed CDNs

Joe Wenjie Jiang*
Princeton University
Princeton, NJ
wenjiej@cs.princeton.edu

Stratis Ioannidis
Technicolor
Palo Alto, CA
stratis.ioannidis@technicolor.com

Laurent Massoulié
INRIA
Paris, France
laurent.massoulie@inria.fr

Fabio Picconi
Technicolor,
Paris, France
fabio.picconi@technicolor.com

ABSTRACT

We consider a content delivery architecture based on geographically dispersed groups of "last-mile" CDN servers, *e.g.*, set-top boxes located within users' homes. These servers may belong to administratively separate domains, such as multiple ISPs. We propose a set of scalable, adaptive mechanisms to jointly manage content replication and request routing within this architecture. Relying on primal-dual methods and fluid-limit techniques, we formally prove the optimality of our design. We further evaluate its performance on both synthetic and trace-driven simulations, based on real BitTorrent traces, and observe a reduction of network costs by more than 50% over traditional mechanisms such as LRU/LFU with closest request routing.

Categories and Subject Descriptors

C.4 [**Computer-Communication Networks**]: [Distributed Systems]

Keywords

Distributed Content Distribution Networks, Content Placement, Request Routing

1. INTRODUCTION

The total Internet traffic per month was already in excess of 10^{19} Bytes in 2011 [1]. Video-on-demand traffic alone is predicted to grow to three times this amount by 2015. Existing content providers such as Youtube and Netflix, which represent a large fraction of today's Internet video traffic, use content delivery networks (CDNs) to replicate, cache, and stream videos at many servers across the world. Nevertheless, the large volumes of traffic exiting the CDN infrastructure incur significant operational costs for the content providers. This state of affairs prompts a rethinking of the current content delivery architecture.

*The author is currently at Google Inc.

A promising evolution from today's approach consists of extending the CDN to the "last mile" of content delivery by incorporating small servers close to the edge of the network. For instance, this approach leverages devices within users' homes, such as set-top boxes or broadband gateways, as advocated by the Nano Datacenter [2] consortium, network attached storage (NAS) such as Boxee [3], or appliances promoted by business initiatives such as AppleTV. Part of or the entire storage and bandwidth capacities of these devices can be leased to a CDN or a content provider. The latter leverage these resources to store content and serve download requests effectively in a managed peer-to-peer fashion.

Such an architecture has considerable advantages for both providers as well as users. First, it harnesses available bandwidth and storage resources of appliances already deployed at users' homes. Second, it enables serving requests locally, *e.g.*, from a device within the same ISP or even within the same neighborhood. In addition to reducing latency, this alleviates CDN server traffic while also reducing cross-traffic among ISPs, thereby significantly decreasing the operational cost of the CDN. In practice, the service operator can make a combined use of the traditional CDN and distributed CDNs to optimize performance and reduce cost simultaneously.

Ideally, one would like to optimize the design of this "diffuse cloud" of nano-servers so that such cross-traffic costs are minimized. There are two degrees of freedom in this design: (a) *content placement*, *i.e.*, identifying what content to cache in each device, and (b) *request routing*, identifying which device is to serve an incoming request for content.

As devices have limited resources, the *storage and bandwidth constraints* of each device need to be taken into account in the above optimization. Also, optimal placement and routing decisions depend on the *demand* for content across users; to reduce traffic costs, it is preferable to cache content closer to where it is most frequently requested.

Nevertheless, this optimization raises several important challenges. First, while traditional CDN design has addressed content caching and request routing separately, a *joint optimization* is required as both decisions impact each other in the case of massively distributed last-mile servers. Second, as the optimization is to be performed over millions of devices, *scalability* is a crucial issue. Third, the system comprises devices dispersed across multiple ISPs and their *heterogeneity*—in terms of bandwidth and storage resources, as well as the relative costs of serving requests—needs to be addressed. In addition, managing many boxes over different ISPs requires a distributed and collaborative solution that does not reveal the internal structure of ISPs. Finally, though optimal placement and routing decisions depend on demand, the latter may be a priori unknown or time-variant. Hence, *adaptive* placement and routing schemes that react to changes in user demand, are preferred.

In this work, we address these challenges by developing a solution for efficient traffic management in next-generation CDNs. We make the following contributions:

- We propose a distributed, adaptive content placement and routing architecture, designed to scale over millions of heterogeneous devices.

- Building upon primal-dual decomposition techniques, we formally establish that our adaptations jointly optimize both content placement and routing, and prove that they minimize CDN traffic costs.

- To establish these results, we characterize the asymptotic loss probability of the *uniform-slot* service assignment policy (determining which device serves a request inside an ISP), a crucial component of our design. We show that although much simpler to implement than prior art, it exhibits the same asymptotic behavior.

- As part of this design, we propose a novel content placement scheme that changes cache contents, and prove that it is order-optimal: it reaches a targeted replication within a constant factor from the optimal number of write operations necessary.

- We conduct simulations driven by BitTorrent traces, with all the associated features of real traffic (burstiness, long-tail distribution), and show that our mechanism reduces network costs by more than half, as compared to traditional solutions based on LRU cache management and nearest-neighbor routing.

The remainder of this paper is organized as follows. We first review related work (Section 2) and describe our problem setup (Section 3). In Section 4 we overview our system architecture. The theoretical guarantees of our adaptation, request routing and content placement algorithms are presented in Sections 5 and 6. Finally, in addition to this theoretical underpinning, our architecture is further validated experimentally in Section 7.

2. RELATED WORK

Peer-assistance of CDNs has been studied from several perspectives. Recent work has shown that it can reduce CDN server traffic [4] and energy consumption [5] by more than 60%. Early research compared the efficiency of prefetching policies for peer-assisted VoD [6], and bandwidth allocation across different P2P swarms [7]. Peer incentivization, *e.g.*, through rebates or service fee reductions, has also been studied [8–10]. Moreover, the use of dedicated home devices as an extension of a CDN's infrastructure has been the model of at least one recent start-up [11] and has been the subject of several recent papers [5,12]. We build and extend upon these works by providing a formal framework for joint placement and routing optimization.

Minimizing cross-traffic is extensively studied in the context of "ISP-friendly" P2P system design and is known to reduce both ISP cross-traffic and download delays [13, 14]. Typically, the selection of download sources is biased towards nearby peers; peer proximity can be inferred either through client-side mechanisms [15] or through a service offered by the ISP [16–18]. In the latter case, the ISP can explicitly recommend which neighbors to download content from by solving an optimization problem that minimizes cross-traffic [18]. In the context of peer-assisted CDNs, an objective that minimizes a weighted sum of cross-traffic and the load on the content server can also be considered [17]. Prior work on ISP-friendliness reduces cross traffic solely by performing *service assignment* to suitable peers. We add a control knob on top of service assignment, namely *content placement*: our optimization selects not only where requests are routed, but also where content is stored.

In the *cooperative caching problem*, clients generate a set of requests for items, that need to be mapped to caches that can serve them; each client/cache pair assignment is associated with an access cost. The goal is to decide how to place items in caches and assign requests to caches that can serve them, so that the total access cost is minimized. The problem is NP-hard, and a 10-approximation algorithm is known [19]. Motivated by CDN topologies, Borst *et al.* [20] obtain lower approximation ratios as well as competitive online algorithms for the case where cache costs are determined by weights in a star graph. A polynomial algorithm is known in the case where caches are organized in a hierarchical topology and the replica accessed is always the nearest replica [21]. We significantly depart from the above studies by explicitly dealing with bandwidth constraints, assuming a stochastic demand for items and proposing an adaptive, distributed algorithm for joint content placement and service assignment among multiple *classes* of identical caches.

Finally, recent work [22–24] has considered cache management specifically in the context of P2P VoD systems, and is in this sense close to our work. However, the heterogeneous (*e.g.*, across multiple ISPs) aspect of our system is not present in any of the above works. As in [24], we capture box service behavior through a loss model; our Thm. 2 extends their analysis, by establishing that a simple service assignment policy (the "uniform slot" policy), has asymptotically the same behavior as the one proposed in [24] (see also Section 6.1).

3. PROBLEM FORMULATION

In the system we consider, a content provider such as YouTube or Netflix delivers content (*e.g.*, videos or music) to home users subscribing to its services. Users access the service through devices installed at their home, such as set-top boxes (providing common Internet connectivity and limited storage) or network-attached storage (NAS) devices. Part of the storage and upload capacities of these boxes is leased to the content provider. The latter uses these resources to serve requests for its content, alleviating the load on its CDN.

Users are geographically dispersed across ISPs. Serving them incurs a cost depending on the cross-traffic they generate. The content provider's goal is to determine (a) where to cache content and (b) how to serve user requests for content, in order to minimize this cost. The remainder of this section formalizes our setup; our key notation is in Table 1.

3.1 Box Classes

We represent the users' home devices by a set $\mathcal{B}$, the set of *boxes*, where $B = |\mathcal{B}|$. We partition $\mathcal{B}$ into D classes $\mathcal{B}^d$, $d \in \mathcal{D} = \{1, \ldots, D\}$, with size $B^d = |\mathcal{B}^d|$. Such partitioning may correspond, *e.g.*, to grouping together boxes managed by the

s	Existing CDN infrastructure.
$\mathcal{B}$	Set of all boxes in the system.
$\mathcal{C}$	Content catalog.
$\mathcal{D}$	Set of all set-of-box classes.
$\mathcal{B}^d$	Boxes in class $d \in \mathcal{D}$.
M^d	Storage capacity of boxes in $\mathcal{B}^d$.
U^d	Upload capacity of boxes in $\mathcal{B}^d$.
$\mathcal{F}_b$	Cache content of box b.
p_c^d	Replication ratio of content c in $\mathcal{B}^d$.
λ_c^d	Request rate for content $c \in \mathcal{C}$ in $\mathcal{B}^d$.
$w^{dd'}$	Cost of a transfer from $d' \in \mathcal{D} \cup \{s\}$ to $d \in \mathcal{D}$.
$r_c^{dd'}$	Rate of requests for c forwarded from $d \in \mathcal{D}$ to $d' \in \mathcal{D} \cup \{s\}$.
$r_c^{\cdot d}$	Aggregate rate of incoming requests for c to $d \in \mathcal{D}$.
R^d	$R^d = B^d U^d$, the total upload capacity in class d.

Table 1: Summary of key notation

same ISP. Different levels of aggregation or granularity may be used: for example, each class may comprise boxes within the same city or even the same city block. Throughout the text, we use the index d for a class in $\mathcal{D}$ and the index s (as in "server") to indicate the existing CDN infrastructure.

Through the CDN, boxes gain access to the provider's content collection, such as, *e.g.*, movies, clips or shows. We denote this collection by $\mathcal{C}$ and call it the *content catalog*. We assume that items in $\mathcal{C}$ have identical size—if not, the original content can be partitioned into fixed-size chunks, and $\mathcal{C}$ viewed as a collection of chunks. Classes are heterogeneous: storage and bandwidth capacities as well as traffic costs associated with serving requests differ across classes. Nevertheless, as described below, boxes within the same class have the same capacities and incur the same costs.

3.2 Storage and Bandwidth Capacity

Part of the boxes' storage is allocated to and managed by the CDN. Each box in $\mathcal{B}^d$ has M^d storage "slots", used by the CDN to store content items. We call M^d the *storage capacity* of boxes in $\mathcal{B}^d$. For each $b \in \mathcal{B}^d$, we denote the set of items cached in box b by $\mathcal{F}_b \subset \mathcal{C}$, where $|\mathcal{F}_b| = M^d$. Note that $\mathcal{F}_b$ is determined by the CDN, not the user. Users may store (or delete at will) content they retrieve in private storage devices, or even at the spare storage of their box, but such replicas are not managed or shared by the CDN.

Boxes can serve incoming requests for content they store in this designated space. We model this service behavior through a *loss model* [25], rather than a queueing model. A box in $\mathcal{B}^d$ can upload at most U^d content items concurrently, each at a fixed rate. We refer to U^d as the *upload capacity* of boxes in class d. Alternatively, each box has U^d upload "slots": if a box receives a request for a content it stores and has a free upload slot, this slot is used to serve the request and upload the requested content. The service time, *i.e.*, the duration of an upload, is assumed to be exponentially distributed with one-unit mean. Slots remain busy until the upload terminates, at which point they become free again.

The use of a loss model ensures that incoming requests are immediately served by a box at a guaranteed rate and no queuing delays are incurred. Moreover, most of today's content services, such as video streaming, require a constant bit-rate and do not consume additional bandwidth, so partitioning uplink bandwidth into "slots" makes sense.

3.3 Request Load

Users (boxes) generate content requests at varying rates across different classes. We model requests for content c generated by each box $b \in \mathcal{B}^d$ through a Poisson process with rate $\tilde{\lambda}_c^d$. Hence, the aggregate request process for c in class d is also Poisson with rate $\lambda_c^d = \tilde{\lambda}_c^d B^d$, which scales proportionally to the class size. When a box $b \in \mathcal{B}^d$ storing $c \in \mathcal{C}$ (*i.e.*, $c \in \mathcal{F}_b$) generates a request for c, it is served by the local cache—no downloading is necessary. Otherwise, the request must be served by either the CDN's pre-existing infrastructure or some other box in $\mathcal{B}$.

Though our analysis assumes that the request generation process is stationary and Poisson (Thms. 1-3) we relax this assumption in Section 7, evaluating our system over requests of time-varying intensity generated by real-life P2P users.

3.4 Minimizing Traffic Costs

Serving a user request from class d using either the existing CDN infrastructure or another class d' requires transferring content across the class boundaries. In general, cross-traffic costs are dictated by the transit agreements between peering ISPs and may vary from one class to the next. As such, we denote by $w^{dd'}$, $d, d' \in \mathcal{D}$, the traffic cost for serving a request from class d by a box in class d'. Similarly, we denote by w^{ds} as the traffic cost of serving a request from class d by the CDN's existing infrastructure.

The content provider that manages the boxes pays incurred cross-traffic costs to ISPs. Hence, it is in its interest to minimize such costs. In particular, the service provider needs to determine (a) the content $\mathcal{F}_b$ placed in each box b, and (b) where the request generated by each box should be directed to, so that its aggregate traffic costs are minimized.

Solving this problem over millions of devices, while satisfying the constraints imposed by the limited storage and bandwidth capacities at each box, poses a significant scalability challenge. Further, deciding where to place content and how to route requests is, in general, a computationally intractable combinatorial problem [19]. In addition, managing boxes from different ISPs raises the need for a distributed solution that does not require the ISPs to reveal their internal structure to each other. Finally, the optimal placement and routing scheme depends on the demands λ_c^d; these may dynamic and a priori unknown to the CDN. As such, an adaptive scheme, that measures and reacts to user demand, is preferable. We present a system design that addresses these challenges in the next section.

4. SYSTEM ARCHITECTURE

To address the challenges above, we propose a distributed, adaptive scheme for joint placement and routing. Through a combination of asymptotic results, we show our design is optimal, in the sense that it minimizes the CDN's aggregate traffic cost when the number of boxes is large.

4.1 Overview

Rather than centralizing the management of the distributed CDN, our solution delegates management to one device per class, termed the *class tracker*. Class trackers are deployed by the content provider, either as separate servers or as designated boxes within each class. They manage (a) content placement within their classes, (b) the routing of requests either generated or served by boxes in their classes.

Each tracker has a full view of the state of boxes within its class, knowing, *e.g.*, the contents of each box's cache and the number of its free upload slots. Nevertheless, the tracker does not have access to the same information about boxes in other classes. In fact, its knowledge about the state in other classes is limited to lightweight *congestion signals* exchanged periodically between trackers.

Overall, trackers perform the following operations:

Adaptation. For each content $c \in \mathcal{C}$, the tracker maintains the desirable *replication ratio* p_c^d, *i.e.*, the fraction of boxes in the class that store c. In addition, it also maintains the desirable *forwarding rates* $r_c^{dd'}$, $d' \in \mathcal{D} \cup \{s\}$, which correspond to the rate of outgoing requests for c, forwarded to other class trackers as well as the CDN infrastructure (denoted by s). These variables are stored locally by the tracker and updated periodically, at fixed time intervals (*e.g.*, once every day). The updated values are used as inputs to the tracker's placement and routing algorithms within the next adaptation round. Updating these variables allows the trackers to adapt both their placement and routing decisions, in a way that the system reaches a global objective (namely, the minimization of aggregate costs).

Content Placement. At the termination of each adaptation round, after deciding the replication ratios p_c^d of each content item in class d, the tracker allocates the content items to boxes: for each box $b \in \mathcal{B}^d$, it determines $\mathcal{F}_b$ in a manner so that the fraction of boxes storing c is indeed p_c^d.

Request Routing. Trackers are responsible for routing requests either generated or served by boxes in their classes. We separate the routing of requests into two phases: *request forwarding* and *service assignment.* Request forwarding determines to which class a request generated by a local box is forwarded, so that the desirable forwarding rates $r_c^{dd'}$ are maintained. Upon receiving a request, the tracker of the class selects a box within its class to serve the request; we refer to this selection as service assignment.

In the remainder of this section, we formally define the optimization performed by the tracker through adaptation and describe our request routing algorithm in detail. We also outline our distributed adaptation and content placement algorithms; their full specification is in Sec. 5 and 6.

4.2 Tracker Information

As stated above, each class tracker has a complete view of the current state of every box inside its own class. In particular, it knows (a) which content items are stored in each box, and (b) how many free upload slots it has. The trackers also collect traffic statistics: the class d tracker maintains estimates of λ_c^d, $c \in \mathcal{C}$, the rate with which requests for content c are generated within the class. It also maintains estimates of the *incoming* rate of requests for content c in class d. All the above are measured and maintained locally at the tracker; this is possible precisely because it manages both content placement and request routing within its class.

Nevertheless, trackers are *a-priori* unaware the states of boxes in other classes: they learn about congestion in other classes through the exchange of appropriate light-weight congestion signals, at the end of each adaptation round.

4.3 Replication and Forwarding Policies

In addition to the above information pertaining to their classes, trackers maintain $|\mathcal{C}|$ local variables $p_c^d \in [0,1]$, for $c \in \mathcal{C}, d \in \mathcal{D}$. We call p_c^d the *replication ratio* of item c in class d, and the vector $\boldsymbol{p}^d = [p_c^d]_{c\in\mathcal{C}}$ the *replication policy* of class d. At any point in time, the replication ratios satisfy:

$$p_c^d = \textstyle\sum_{b\in\mathcal{B}^d} \mathbb{1}_{c\in\mathcal{F}_b}/B^d, \quad \forall c \in \mathcal{C}, d \in \mathcal{D}, \tag{1}$$

i.e., the replication ratio p_c^d equals the fraction of boxes in $\mathcal{B}^d$ that store content $c \in \mathcal{C}$. Also, by summing (1) in terms of c, it is easy to see that, when all caches are full,

$$\textstyle\sum_{c\in\mathcal{C}} p_c^d = M^d, \quad \forall d \in \mathcal{D}. \tag{2}$$

The replication policy of a tracker serves as an input to its content placement algorithm. That is, the tracker updates it replication policy (or, its desired replication ratios) at the end of every adaptation round. Subsequently, the replication policy is used to determine the placement of content to boxes in $\mathcal{B}^d$, so that (1) is indeed satisfied.

In addition to its replication policy, the tracker maintains $(|\mathcal{D}|+1)\times|\mathcal{C}|$ additional local variables

$$r_c^{dd'}, \quad d' \in \mathcal{D}\cup\{s\}, c \in \mathcal{C}.$$

We call these the *forwarding rates* of class d, and the vector $\boldsymbol{r}^d = [r_c^{dd'}]_{d'\in\mathcal{D}\cup\{s\}\mathcal{C}}$ of these values the *forwarding policy* of d. At any point in time each variable $r_c^{dd'} \in \mathbb{R}_+$, for $d' \in \mathcal{D}$, equals the rate of requests for content c that the tracker forwards from class d to d'. Similarly, each variable $r_c^{ds} \in \mathbb{R}_+$ equals the rate of requests for c forwarded by the tracker directly to the CDN's existing infrastructure. The forwarding policies satisfy the equalities:

$$r_c^{ds} + \textstyle\sum_{d'\in\mathcal{D}} r_c^{dd'} = \lambda_c^d(1-p_c^d), \quad \forall c \in \mathcal{C}, d \in \mathcal{D}, \tag{3}$$

i.e., requests not immediately served by local caches are forwarded to the CDN or a box in another class.

The tracker also updates its forwarding policy at the end of an adaptation round. The updated values are subsequently used as inputs to the tracker's request forwarding algorithm for the next round. In particular, the tracker implements a routing scheme that ensures that the rate of requests for item c forwarded to $d' \in \mathcal{C}\cup\{s\}$ is precisely $r_c^{dd'}$.

4.4 Request Routing and Loss Probabilities

As mentioned above, the routing of requests in our scheme consists of two phases, *request forwarding* and *service assignment.* We describe these in detail below.

Request Forwarding. In the request forwarding phase, a box $b \in \mathcal{B}^d$ generating a request for an item $c \in \mathcal{C}$ first checks if it already stores c, *i.e.*, $c \in F_b$. If so, the request is served immediately and no downloading is necessary. If not, the box contacts the class tracker; the latter determines whether the request should be forwarded to (a) another box within the class, (b) a box in another class, or (c) served directly by the CDN's infrastructure. If the tracker determines that the request is to be forwarded to class d' (case (b)), it routes the request to the tracker managing this class.

To select among these 3 outcomes, the tracker uses $\boldsymbol{r}^d$ as follows: it forwards a request to $d' \in \mathcal{D}\cup\{s\}$ with a probability proportional to $r_c^{dd'}$. As a result, provided that (3) is satisfied, requests forwarded from class d to d' form independent Poisson processes with rates $r_c^{dd'}$.

Service Assignment. In the service assignment phase, the class d tracker assigns a request for a content c to the box in its class that is to serve it. Requests can be local, *i.e.*, generated by a box in $\mathcal{B}^d$ and deemed to be served locally during the forwarding phase, or external, *i.e.*, generated by a box in a different class d' and forwarded to the class d tracker by the tracker of d'. To assign requests to boxes, the tracker follows a *uniform slot* policy. Under this policy, an incoming request for content c is assigned to a box selected among all boxes currently storing c and having an empty upload slot. Each such box is selected with a probability proportional to the number of its empty slots. Equivalently, the request is matched to a slot selected uniformly from all free upload slots of boxes storing c: for X_b the number of free slots of box $b \in \mathcal{B}^d$, an incoming request for content c is mapped to a slot selected uniformly at random among the $\sum_{b\in\mathcal{B}^d:c\in\mathcal{F}_b} X_b$ slots of boxes that can serve this request.

Loss Probabilities. It is possible that no free upload slots in the class exist when the request for c arrives (*i.e.*, $\sum_{b\in\mathcal{B}^d:c\in\mathcal{F}_b} X_b = 0$). In such a case, a request is re-routed to the CDN's infrastructure. Hence, *not all requests for content* c that arrive at class d are served by boxes in $\mathcal{B}_d$.

Let ν_c^d be the *loss probability* of item c in class d, *i.e.*, the probability that a request for c cannot be served and is re-routed to the infrastructure. We say that requests for item c are served *with high probability* (w.h.p.) in class d, if

$$\lim_{B^d\to\infty} \nu_c^d(B^d) = 0, \tag{4}$$

i.e., as the total number of boxes increases, the probability that a request for content c fails goes to zero. Two necessary constraints (see, *e.g.*, [24]) for (4) to hold in class $d \in \mathcal{D}$ are:

$$\sum_{c\in\mathcal{C}} r_c^{\cdot d} < B^d U^d, \tag{5}$$

$$r_c^{\cdot d} < B^d U^d p_c^d, \quad \forall c \in \mathcal{C}, \tag{6}$$

where $r_c^{\cdot d} = \sum_{d'\in\mathcal{D}} r_c^{d'd}$ is the aggregate request rate for content c received by class d. Constraint (5) states that the aggregate traffic load imposed on class d should not exceed the total upload capacity over all boxes; (6) states that the traffic imposed on d by requests for c should not exceed the total capacity of boxes storing c.

4.5 Content Placement

Our content placement algorithm is presented in detail in Section 6. In summary, the algorithm is executed at the end of every adaptation round. It receives as input the replication policy $\boldsymbol{p}^d$ and generates a placement $\{\mathcal{F}_b\}_{b\in\mathcal{B}^d}$ that is consistent with (1).

Implementing the new placement requires copying new contents to box caches; transferring content incurs traffic costs. Our design ensures these costs are small in two ways. First, policy adaptations are *smooth*, *i.e.*, changes in $\boldsymbol{r}^d, \boldsymbol{p}^d$ are gradual. Second, we implement the new placement by performing *as few changes to box contents as possible.*

Crucially, our placement also satisfies the following property: when uniform slot service assignment is used, *all requests are satisfied w.h.p.* In particular, our placement is such that (5) and (6) are not only necessary but also *sufficient* for (4) to hold (*c.f.* Thms. 2 and 3). As such, our content placement ensures that, asymptotically, almost no requests are re-routed to the CDN.

4.6 Global Optimization

Recall that the cost incurred when a request originating from class d is served by $d' \in \mathcal{D}\cup\{s\}$ is $w^{dd'}$. Moreover, the rate of requests for content c forwarded from d to d' is $r_c^{dd'}$. Hence, the total traffic cost is

$$\sum_{c\in\mathcal{C}}\sum_{d\in\mathcal{D}}\big[w^{ds}r_c^{ds} + \sum_{d'}\big(w^{dd'}r_c^{dd'}(1-\nu_c^{d'}) + w^{ds}r_c^{dd'}\nu_c^{d'}\big)\big].$$

This is because a fraction ν_c^d requests for content c arriving at class d are re-routed to the CDN. In general, this is not a convex function, due to the loss probabilities ν_c^d. However, given that (4) holds, the contribution of these losses becomes negligible for large system sizes. The total system costs can thus be approximated as $\sum_{d\in\mathcal{D}} F^d(\boldsymbol{r}^d)$, where

$$F^d(\boldsymbol{r}^d) = \sum_{c\in\mathcal{C}}\big(w^{ds}r_c^{ds} + \sum_{d'\in\mathcal{D}} w^{dd'}r_c^{dd'}\big) \tag{7}$$

is the total traffic cost generated by class d. Hence, the operator's minimal cost is a solution to the linear program:

GLOBAL

$$\text{Min.} \quad \sum_{d\in\mathcal{D}} F^d(\boldsymbol{r}^d) \tag{8a}$$

$$\text{subj. to} \quad \sum_{c\in\mathcal{C}} p_c^d = M^d, \ \forall d \in \mathcal{D} \tag{8b}$$

$$\sum_{d'\in\mathcal{D}} r_c^{dd'} + r_c^{ds} = \lambda_c^d(1-p_c^d), \ \forall c\in\mathcal{C}, d\in\mathcal{D} \tag{8c}$$

$$\sum_{c\in\mathcal{C}} r_c^{\cdot d} < R^d, \ \forall d \in \mathcal{D} \tag{8d}$$

$$r_c^{\cdot d} < R^d p_c^d, \ \forall c\in\mathcal{C}, d \in \mathcal{D} \tag{8e}$$

$$r_c^{dd'} \geq 0, r_c^{ds} \geq 0, 1 \geq p_c^d \geq 0, \ \forall c \in \mathcal{C}, d, d' \in \mathcal{D}$$

$$\text{var.} \ \boldsymbol{r}^d, \boldsymbol{p}^d, \forall d \in \mathcal{D}$$

where $R^d = B^d U^d$ is the total upload capacity in class d. The objective of this optimization problem is to minimize the total cost incurred by content transfers. Constraints (8b) and (8c) correspond to equations (2) and (3); they state that the full storage capacity of each class is used and that all requests are eventually served, respectively. Constraints (8d) and (8e) correspond to (5) and (6), respectively.

GLOBAL is a linear program in $\boldsymbol{r}^d, \boldsymbol{p}^d$, $d \in \mathcal{D}$. Our distributed, adaptive method for updating the routing and placement policies, is presented in detail in Section 5. It is designed in a way so that (a) trackers measure and adapt their policies to user demand, and (b) policy updates are computed in a distributed fashion, thus scaling well as the number of boxes increases. Most importantly, the policies of our design converge to a solution of **GLOBAL** (see Thm. 1), *i.e.*, our design minimizes aggregate traffic costs.

The formal properties of our design are shown in the next two sections. In Section 5, we specify how trackers update their policies so that they converge to a solution of (8). In Section 6, we characterize content placements under which requests assigned by a uniform slot policy are served *w.h.p.* Moreover, we show that if (8d) and (8e) hold, such a content placement exists, and give an algorithm implementing it.

5. POLICY ADAPTATION

We now present how the trackers solve **GLOBAL** and determine their replication and forwarding policies in a distributed fashion. In short, trackers exchange congestion signals and update $\boldsymbol{p}^d, \boldsymbol{r}^d$ over several rounds. We ensure that both are updated in a smooth fashion, *i.e.*, changes between two rounds are incremental and the system does not oscillate wildly. We proceed by first discussing the challenges in solving (8) in a distributed fashion with classical methods, and then presenting our distributed implementation.

5.1 Standard Dual Decomposition

Consider the partial Lagrangian of (8).

$$\mathcal{L}(\boldsymbol{r},\boldsymbol{p};\boldsymbol{\alpha},\boldsymbol{\beta}) = \textstyle\sum_d F^d(\boldsymbol{r}^d) + \sum_{d'} \beta^{d'}\big(\sum_{c,d} r_c^{dd'} - R^{d'}\big)$$
$$+ \textstyle\sum_{d'}\sum_c \alpha_c^{d'}\big(\sum_d r_c^{dd'} - R^{d'} p_c^{d'}\big)$$

where $\alpha_c^{d'}, \beta^d$ are the dual variables (Lagrange multipliers) associated with the constraints (8d) and (8e), respectively. Observe that $\mathcal{L}$ is separable in the primal variables, *i.e.*, it can be written as $\mathcal{L}(\boldsymbol{r},\boldsymbol{p},\boldsymbol{\alpha},\boldsymbol{\beta}) = \sum_d \mathcal{L}^d(\boldsymbol{r}^d,\boldsymbol{p}^d;\boldsymbol{\alpha},\boldsymbol{\beta})$ where

$$\mathcal{L}^d(\boldsymbol{r}^d,\boldsymbol{p}^d,\boldsymbol{\alpha},\boldsymbol{\beta}) = F^d(\boldsymbol{r}^d) - \beta^d R^d - \textstyle\sum_c \alpha_c^d R^d p_c^d$$
$$+ \textstyle\sum_{d'}\big(\beta^{d'}\sum_c r_c^{dd'} + \sum_c \alpha_c^{d'} r_c^{dd'}\big).$$

This suggests a standard dual decomposition algorithm [26] for solving **GLOBAL**. Recall that a dual decomposition algorithm runs in multiple rounds $t = 0, 1, \ldots$. The class d tracker maintains the primal variables $\boldsymbol{r}^d$, $\boldsymbol{p}^d$, as well as the dual variables $\boldsymbol{\alpha}^d = [\alpha_c^d]_{c\in\mathcal{C}}$, β^d, associated with the coupling constraints (8d) and (8e). At the end of each round, the tracker updates the dual variables α_c^d, β^d, increasing them when the respective constraints (8d) and (8e) are violated or decreasing them when the constraints are loose. Subsequently, each tracker broadcasts its current dual variables with all other trackers. Having all dual variables $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ in the system, the trackers adapt their primal variables, reducing traffic forwarded to congested classes and increasing traffic forwarded to non-congested ones. This can be performed by each tracker solving the linear program:

$$(\boldsymbol{r}^d,\boldsymbol{p}^d)(t+1) = \underset{(\boldsymbol{r}^d,\boldsymbol{p}^d)\in\mathcal{I}^d}{\operatorname{argmin}} \; \mathcal{L}^d(\boldsymbol{r}^d,\boldsymbol{p}^d;\boldsymbol{\alpha}(t),\boldsymbol{\beta}(t)). \qquad (9)$$

where $\mathcal{I}^d$ is the set of pairs $(\boldsymbol{r}^d,\boldsymbol{p}^d)$ defined by (8b) and (8c) as well as the non-negativity constraints. Such adaptations are known to converge to a maximizer of the primal problem *when the functions $\mathcal{L}^d$ are strictly convex* (see, *e.g.*, [26] Section 3.4.2, pp. 229-230). Unfortunately, this is not the case in our setup, as $\mathcal{L}^d$ are linear in $\boldsymbol{r}^d$, $\boldsymbol{p}^d$: convergence to optimal policies does not readily follow. In practice, the lack of strict convexity makes $\boldsymbol{r}^d$, $\boldsymbol{p}^d$ oscillate wildly with every application of (9). This is disastrous: wide oscillations of $\boldsymbol{p}^d$ imply that a large fraction of boxes in $\mathcal{B}^d$ need to change their content in each round. This is both impractical and costly; ideally, we would like each round to change the contents of each class smoothly, so that the cost of implementing these changes is negligible.

5.2 A Smooth Distributed Implementation

To address these issues, we use an interior point method that deals with the lack of strict convexity called *the method of multipliers* [26]. Applied to **GLOBAL** this implementation yields the algorithm summarized in Fig. 1. The following theorem, proved in App. A, follows from the analysis in [26] and establishes that this algorithm indeed solves (8):

Theorem 1. *Assume that the tracker in class d correctly estimates $\lambda_c^d, r_c^{\cdot d}$ in each round, and that $\{\theta(t)\}_{t\in\mathbb{N}}$ is a non-decreasing sequence of non-negative numbers. Then, under the adaptation algorithm in Fig. 1, $\boldsymbol{r}^d(t), \boldsymbol{p}^d(t)$ converge to an optimal solution of* (8).

Crucially, the algorithm in Fig. 1 performs smooth adaptations of $\boldsymbol{r}^d, \boldsymbol{p}^d$. Though the theorem assumes that trackers

```
Tracker $d$ at the end of round $t$:
  Obtain estimates of $\lambda_c^d$, $r_c^{\cdot d}$, $c \in \mathcal{C}$.
  // Update dual variables
  $s_{tot}^d \leftarrow \frac{1}{|\mathcal{D}|}\left(\sum_{c\in\mathcal{C}} r_c^{\cdot d} + y^d - R^d\right)$
  $\beta^d \leftarrow \beta_c^d + \theta s_{tot}^d$
  for each content $c$
    $s_c^d \leftarrow \frac{1}{|\mathcal{D}|}\left(r_c^{\cdot d} + z_c^d - R^d p_c^d\right)$
    $\alpha_c^d \leftarrow \alpha_c^d + \theta s_c^d$
  end for
  Broadcast $(\boldsymbol{\alpha}^d, \beta^d, \boldsymbol{s}^d, s_{tot}^d)$ to other trackers $d' \in \mathcal{D}$
  Receive dual variables from all other trackers $d' \in \mathcal{D}$
// Update primal variables
  $(\boldsymbol{r}^d, \boldsymbol{p}^d, \boldsymbol{z}^d, y^d) \leftarrow \operatorname{argmin}_{\mathcal{I}^d} \mathbf{LOCAL}^d(\boldsymbol{r}^d, \boldsymbol{p}^d, \boldsymbol{z}^d, y^d, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{s}, s_{tot})$
```

Figure 1: Decentralized solution to the global problem GLOBAL.

correctly estimate the request rates λ_c^d, which are stationary, we relax both assumptions in Section 7, evaluating our adaptive approach under real-life, time-varying traffic.

We describe the operations performed and messages exchanged by each tracker below. The class d tracker maintains $\boldsymbol{r}^d(t), \boldsymbol{p}^d(t), \boldsymbol{\alpha}^d(t), \beta^d(t)$, the primal and dual variables of (8), as well as the *slack variables* y^d, $\boldsymbol{z}^d = [z_c^d]_{c\in\mathcal{C}}$, resulting from converting of (8d) and (8e) to equality constraints:

$$\textstyle\sum_{c\in C} r_c^{\cdot d} + y^d = R^d, \qquad \forall d \in \mathcal{D} \qquad (10a)$$
$$r_c^{\cdot d} + z_c^d = R^d p_c^d, \qquad \forall c \in \mathcal{C}, d' \in \mathcal{D} \qquad (10b)$$
$$y^d \geq 0,\; z_c^d \geq 0, \qquad \forall c \in \mathcal{C}, d \in \mathcal{D} \qquad (10c)$$

In addition, for every $c \in \mathcal{C}$, the tracker maintains an estimate of λ_c^d, *i.e.*, the request rate of c from boxes within its own class, as well as an estimate of $r_c^{\cdot d}$,*i.e.*, the request rate for content c served by boxes in $\mathcal{B}^d$. These can be estimated through appropriate counters or through more sophisticated moving-average methods (such as, *e.g.*, EWMA).

Using these estimates, the primal and dual variables are updated as follows. At the end of round t, the tracker in class d uses the estimates of $r_c^{\cdot d}$ to see whether constraints (10a) and (10b) are violated or not. In particular, the tracker computes the quantities:

$$s_{tot}^d(t) = \left(\textstyle\sum_c r_c^{\cdot d}(t) + y^d(t) - R^d\right)/|\mathcal{D}|$$
$$s_c^d(t) = \left(r_c^{\cdot d}(t) + z_c^d(t) - R^d p_c^d(t)\right)/|\mathcal{D}|, \quad \forall c \in \mathcal{C}$$

and updates the dual variables as follows:

$$\beta^d(t) = \beta^d(t-1) + \theta(t) s_{tot}^d(t)$$
$$\alpha_c^d(t) = \alpha_c^d(t-1) + \theta(t) s_c^d(t), \quad \forall c \in \mathcal{C}$$

where $\{\theta(t)\}_{t\in\mathbb{N}}$ are positive and non-decreasing. Subsequently, the tracker broadcasts to *every other tracker in $\mathcal{D}$* its congestion signals $\boldsymbol{\alpha}^d(t), \beta^d(t), \boldsymbol{s}^d(t), s_{tot}^d(t)$. This entails the exchange of $2|\mathcal{D}|(|\mathcal{D}|+1)$ values, in total.

For any $d, d' \in \mathcal{D}$, let $G_{tot}^{dd'}(\boldsymbol{r}^d, y^d) = \sum_c r_c^{dd'} + \mathbb{1}_{d=d'} y^d$, and $G_c^{dd'}(\boldsymbol{r}^d, \boldsymbol{p}^d, \boldsymbol{z}^d) = r_c^{dd'} + \mathbb{1}_{d=d'}(z_c^d - R^d p_c^d)$. Intuitively, these capture the "contribution" of the primal variables of class d to the constraints (10a) and (10b) of class d'. After the tracker in class d has received all the messages sent by other trackers, it solves the following quadratic program:

$$\mathbf{LOCAL}^d(\boldsymbol{r}^d(t),\boldsymbol{p}^d(t),\boldsymbol{z}^d(t),y^d(t),\boldsymbol{\alpha}(t),\boldsymbol{\beta}(t),\boldsymbol{s}(t),\boldsymbol{s}_{tot}(t))$$

$$\begin{aligned}
\text{Min. } & F^d(\boldsymbol{r}^d)+\sum_{d'}\beta^{d'}(t)G^{dd'}_{tot}(\boldsymbol{r}^d,y^d)\\
&+\sum_{d',c}\alpha^{d'}_c(t)G^{dd'}_c(\boldsymbol{r}^d,\boldsymbol{p}^d,\boldsymbol{z}^d)\\
&+\frac{\theta(t)}{2}\sum_{d'}\Bigg[\Big(G^{dd'}_{tot}\big(\boldsymbol{r}^d-\boldsymbol{r}^d(t),y^d-y^d(t)\big)+s^{d'}_{tot}(t)\Big)^2\\
&+\sum_c\Big(G^{dd'}_c\big(\boldsymbol{r}^d-\boldsymbol{r}^d(t),\boldsymbol{p}^d-\boldsymbol{p}^d(t),\boldsymbol{z}^d-\boldsymbol{z}^d(t)\big)+s^{d'}_c(t)\Big)^2\Bigg]\\
\text{s.t. } & (\boldsymbol{r}^d,\boldsymbol{p}^d,\boldsymbol{z}^d,y^d)\in\mathcal{J}^d,\ \forall d\in\mathcal{D}\\
\text{var } & \boldsymbol{r}^d,\boldsymbol{p}^d,\boldsymbol{z}^d,y^d,\quad d\in\mathcal{D}
\end{aligned}$$

where $\mathcal{J}^d$ is the set of quadruplets $(\boldsymbol{r}^d,\boldsymbol{p}^d,\boldsymbol{y}^d,z^d)$ defined by (8b) and (8c) as well as the non-negativity constraints. $\mathbf{LOCAL}^d$ thus receives as input *all* the dual variables $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, the congestion variables $\boldsymbol{s}^d$, $\boldsymbol{s}^d_{tot}$, *as well as* all the local primal variables at round t. The last four are included in the quadratic terms appearing in the objective function, and ensure the smoothness of the changes to the primal variables from one round to the next.

6. CONTENT PLACEMENT

The previous section establishes that, through an appropriate exchange of congestion signals, trackers can solve (8) in a distributed fashion. Nevertheless, the objective (8a) is only an approximation of the actual traffic; this is because requests reaching a class may be "dropped" and redirected to the CDN infrastructure. In this section, we describe our content placement scheme and show that it ensures that all incoming requests are satisfied *w.h.p.*.

6.1 Conditions for Non-Redirection

We begin by establishing the necessary and sufficient conditions for requests to succeed *w.h.p.*, when the trackers implement the uniform slot service assignment.

Consider a collection of contents $\mathcal{F}\subset\mathcal{C}$ such that $|\mathcal{F}|=M^d$. Let $\mathcal{B}^d_{\mathcal{F}}=\{b\in\mathcal{B}^d:\mathcal{F}_b=\mathcal{F}\}$ be the set of boxes in the class that store exactly $\mathcal{F}$. These sets partition $\mathcal{B}^d$ into sub-classes, each comprising boxes that store identical contents. Let the number of boxes $B=|\mathcal{B}|$ go to infinity, while scaling both the request arrival rates $r^{\cdot d}_c$ and the size of the subclasses $B^d_{\mathcal{F}}=|\mathcal{B}^d_{\mathcal{F}}|$ proportionally to B. That is, the quantities $r^{\cdot d}_c/B$, $B^d_{\mathcal{F}}/B$ are constants that do not depend on B as the latter increases. This scaling is consistent with our design: as B increases, the aggregate content demand and the storage and upload capacities grow proportionally with B. The following theorem, proved in App. B, characterizes the conditions under requests succeed *w.h.p.*

Theorem 2. *Assume that requests are assigned according to the uniform slot policy. Then, requests for every content $c\in\mathcal{C}$ are served* w.h.p. *if and only if*

$$\textstyle\sum_{c\in A}r^{\cdot d}_c<\sum_{\mathcal{F}:\mathcal{F}\cap A\neq\emptyset}B^d_{\mathcal{F}}U^d,\quad \textit{for all } A\subseteq\mathcal{C},\qquad(11)$$

Condition (11) stipulates that for any set of items $A\subseteq\mathcal{C}$ the arrival rate of requests for these items does not exceed the total upload capacity of class d boxes storing these items.

It is relatively straightforward to see that (11) is *necessary* for (4) to hold (see [24]). It has recently shown that it is also *sufficient* when the service assignment policy used is the so-called *repacking policy* [24]. At the arrival of a request, repacking re-assigns requests already served in boxes in the system in order to accommodate this request. Performing this "repacking" requires finding a maximum matching in a bipartite graph of $2B^dU^d$ nodes. Our uniform slot policy is thus easier to implement than repacking; moreover, Thm. 2 establishes that, despite its simplicity, it exhibits the same asymptotic performance.

The theorem implies that, to serve all requests in a class *w.h.p.*, the content placement should be such that condition (11) is satisfied. Unfortunately, this condition consists of a number of inequalities that is exponential in the catalog size $|\mathcal{C}|$, and is not a priori clear how to construct a content placement scheme. We address this in the next section.

6.2 Placement Algorithm

In this section, we show that if the conditions (8d) and (8e) of **GLOBAL** hold, there exists a simple content placement scheme that satisfies (11). This has the following immediate implications. First, it simplifies our design, as we only need to ensure that the $O(|\mathcal{D}||\mathcal{C}|)$ constraints (8d) and (8e) hold, rather than the exponentially many constraints in (11); indeed it is only these constraints that our adaptation scheme of Section 5 takes into account. Second, by Thm. 2, implementing the placement in this section along with a uniform slot assignment ensures that all requests are served *w.h.p.*

Below, we first describe this placement scheme, *i.e.* the mapping of contents to boxes caches, that satisfies (11). We then present an algorithm that, at each round, reshuffles cache contents in the class to reach this placement with as few item transfers as possible.

Designated Slot Placement. We now show that if (8d) and (8e) hold, there exists a simple placement scheme—*i.e.*, a set of cache contents $\{F_b\}_{b\in\mathcal{B}^d}$—that satisfies (11).

For every box $b\in\mathcal{B}^d$, we identify a special storage slot which we call the *designated slot.* We denote the content of this slot by D_b and the remaining contents of b by $L_b=\mathcal{F}_b\setminus\{D_b\}$. For all $c\in\mathcal{C}$, let $\mathcal{E}^d_c=\{b\in\mathcal{B}^d:D_b=c\}$ be the set of boxes storing c in their designated slot. The following lemma implies that if a sufficient number boxes store c in their designated slot, then (11) is satisfied.

Lemma 1. *If $|\mathcal{E}^d_c|>r^{\cdot d}_c/U^d$ then* (11) *holds.*

Proof. As $\mathcal{E}^d_c$ are disjoint, we have $\sum_{\mathcal{F}:\mathcal{F}\cap A\neq\emptyset}B^d_{\mathcal{F}}U^d=\sum_{b\in\mathcal{B}^d:\mathcal{F}_b\cap A\neq\emptyset}U^d=\sum_{b\in\mathcal{B}^d:D_b\in A}U^d+\sum_{b\in\mathcal{B}^d:D_b\notin A}U^d\geq\sum_{c\in A}|\mathcal{E}^d_c|U^d>\sum_{c\in A}r^{\cdot d}_c$, for all $A\subseteq\mathcal{C}$. □

Hence, to ensure that (11) is satisfied, it suffices that at least $r^{\cdot d}_c/U^d$ boxes store c in their designated slot. We call such a placement scheme a *designated slot placement.* On the other hand, the fraction of boxes that store c in *any* slot must not exceed p^d_c. The following lemma states that is possible to place contents in each designated slot to ensure that both constraints are satisfied when (8d) and (8e) hold:

Lemma 2. *Given a class d, consider $r^{\cdot d}_c$ and p^d_c, $c\in\mathcal{C}$, for which* (8d) *and* (8e) *hold. There exist $q^d_c\in[0,1]$, $c\in\mathcal{C}$, such that $\sum_c q^d_c=1$ and*

$$0\leq r^{\cdot d}_c/B^dU^d<q^d_c\leq p^d_c\leq 1,\ \forall c\in\mathcal{C}.\qquad(12)$$

Moreover, such q^d_c can be computed in $O(|\mathcal{C}|\log|\mathcal{C}|)$ time.

The proof can be found in App. C. In summary, if (8d) and (8e) hold, ensuring that requests for all contents are served

```
**Input:** Initial placement $\{F_b\}_{b\in\mathcal{B}}$ and target ratios $q'_c$, $p'_c$
Let $A^+ := \{c \in \mathcal{C} : q_c > q'_c\}$, $A^- := \{c :\in \mathcal{C} : q_c < q'_c\}$;
**while** there exists $b \in \mathcal{B}$ s.t. $D_b \in A^+$ and $L_b \cap A^- \neq \emptyset$
  Pick $c \in L_b \cap A^-$, and swap it locally with the content of $D_b$.
  Update $\boldsymbol{q}$, $\boldsymbol{\pi}$, $A^+$, $A^-$ accordingly
**while** there exists $b \in \mathcal{B}$ s.t. $D_b \in A^+$ and $L_b \cap A^- = \emptyset$
  Pick $c \in A^-$ and place $c$ in the designated slot $D_b$;
  Update $\boldsymbol{q}$, $\boldsymbol{\pi}$, $A^+$, $A^-$ accordingly
Let $C^+ := \{c : \pi_c > \pi'_c\}$; $C^+ := \{c : \pi_c < \pi'_c\}$;
$C^0 := \mathcal{C} \setminus (C^+ \cup C^-)$.
Let $G := \{\ b \in \mathcal{B}$ s.t. $C_+ \cap L_b \neq \emptyset$ and $C_- \setminus (D_b \cup L_b) \neq \emptyset\ \}$;
**while** $(G \neq \emptyset)$ **or** (there exists $c \in C^-$ s.t. $(\pi_c - \pi'_c)B \geq 2$)
  **if** $(G \neq \emptyset)$ **then**
    Pick any $b \in G$
    Replace some $c \in C_+ \cap L_b$ with some $c' \in C_- \setminus (D_b \cup L_b)$;
  **else**
    Pick $c \in C^-$ s.t. $(\pi_c - \pi'_c)B \geq 2$.
    Find a box $b$ that does not store $c$.
    Pick $c' \in C_0 \cap L_b$ and replace $c'$ with $c$.
  update $G$, $\boldsymbol{\pi}$, $C^+$, $C^-$, $C^0$ accordingly.
```

Figure 2: Placement Algorithm

w.h.p. in class d is achieved constructing a designated slot placement. Such a placement stores content c in the designated slot of at least $q_c^d B^d$ boxes, where $q_c^d B^d$ are determined as in Lemma 2; the remaining slots are used to achieve an overall replication ratio of p_c^d within the class. Below, we describe an algorithm that, given ratios q_c^d and p_c^d, places content in class d in a way that these ratios are satisfied. For simplicity, we drop the superscript d in the remainder of this section, referring to content placement in a single class.

Constructing a Designated Slot Placement. We now describe how to change cache contents at the end of each adaptation round. The tracker is aware of the initial content placement $\{\mathcal{F}_b\}_{b\in\mathcal{B}}$ over B boxes in set $\mathcal{B}$, prior to the adaptation, as well as the target (*i.e.*, adapted) replication ratios p'_c and q'_c, $c \in \mathcal{C}$, satisfying (12); the former are given by the adaptation algorithm, and the latter by Lemma (2). The placement algorithm, outlined in Fig. 2, receives these as inputs and outputs a new content placement $\{\mathcal{F}'_b\}_{b\in\mathcal{B}}$ in which $q'_c B$ boxes store c in their designated slot, while *approximately* $p'_c B$ boxes store c overall. Crucially, the placement requires *as few cache changes as possible.*

We assume that $q'_c B$ and $p'_c B$ are integers—for large B, this is a good approximation. Let q_c, p_c be the corresponding designated slot and overall fractions in the input placement $\{F_b\}_{b\in\mathcal{B}}$. Let $\pi_c = p_c - q_c$, $\pi'_c = p'_c - q'_c$. A lower bound on the cache modification operations needed to attain the target replication ratios q'_c and p'_c is given by $B(\alpha + \beta)/2$, where $\alpha = \sum_c |q_c - q'_c|$, $\beta = \sum_c |\pi_c - \pi'_c|$. We also express the number of operations performed in terms of these quantities. The complexity and correctness of the algorithm are established in the following theorem, whose proof is in App. D:

Theorem 3. *The content placement algorithm in Fig. 2 leads to a content replication $\{F'_b\}_{b\in\mathcal{B}}$ in which exactly $q'_c B$ boxes store c in their designated slot, and $p''_c B$ boxes store c overall, where $\sum_c |p'_c - p''_c| B < 2M$, and $|p'_c - p''_c| B \leq 1$, for all $c \in \mathcal{C}$. The total number of write operations is at most $B[\alpha + (M-1)(\alpha + \beta)]/2$.*

In summary, our algorithm produces a placement in which at most $2M$ items are either under or over-replicated, each by *only one* replica. Most importantly, the placement is achieved with at most $O(B(\alpha + \beta))$ write operations, which is order-optimal. If replication ratios change gradually (and, thus, α and β are small), as ensured by our policy adaptation, the algorithm does not perform a large number of cache changes. We describe the algorithm in more detail below.

The algorithm proceeds in three phases. In the first phase, the algorithm modifies the designated slots to reach the desired ratios $\boldsymbol{q}'$. To do so, the algorithm picks any over-replicated content c in set $A_+ = \{c : q_c > q'_c\}$. For any user holding c in its designated slot, it checks whether it holds in its normal slots an under-replicated content $c' \in A_- = \{c : q_c < q'_c\}$. If such content exists, it renames the corresponding slot as "designated" and the slot holding c as "normal". This is repeated until an under-replicated content c' cannot be found within the normal cache slots of boxes storing some $c \in A_+$. If there still are over-replicated items in A_+, some $c' \in A_-$ is selected arbitrarily and overwrites c within the designated slot. At the end of this phase, the replication rates within the designated slots have reached their target $B\vec{q'}$, and the resulting caches are free of duplicate copies. Also, after these operations, the intermediate replication rates π''_c within the normal cache slots verify $|B\pi_c - B\pi''_c| \leq |Bq_c - Bq'_c|$.

In the second phase, the algorithm begins transforming these intermediate replication rates π''_c into π'_c. To this end, we distinguish contents c that are over-replicated, under-replicated and perfectly replicated by introducing $C_+ = \{c\colon \pi_c > \pi'_c\}$, $C_- = \{c\colon \pi_c < \pi'_c\}$, and $C_0 = \{c\colon \pi_c = \pi'_c\}$.

$C_+ = \{c : \pi_c > \pi'_c\}, C_- = \{c : \pi_c < \pi'_c\}, C_0 = \{c : \pi_c = \pi'_c\}$.

For any box b, if there exists $c \in C_+ \cap L_b$, and $c' \in C_- \setminus (D_b \cup L_b)$, the algorithm replaces c by c' within L_b. We call the corresponding operation a *greedy reduction.* Greedy reductions are repeated until the algorithm arrives at a configuration where no such changes are possible; this terminates the second phase. At that point, for any box b such that $C_+ \cap L_b$ is not empty, necessarily $C_- \subset (L_b \cup D_b)$. Hence, the size of C_- is at most $M - 1$. If any of the elements in C_- is under replicated by at least two replicas, the algorithm enters its third phase. In this phase, the algorithm picks some content c' that is under-replicated by at least 2 replicas, and finds a user b which does not hold c', *i.e.* $c' \in C_- \setminus (D_b \cup L_b)$. It also selects some content c within $C_0 \cap L_b$: such content must exist, since $|C_-| \leq M - 1$, and $C_- \cap L_b \subset C_- \setminus \{c'\}$ has size strictly less than $M - 1$, the size of L_b; the remaining content c must belong to C_0 since otherwise we could have performed a greedy reduction.

The algorithm then replaces content c by content c'. We call this operation a *switch.* This augments the size of set C_-: indeed content c is now under-replicated (one replica missing). The algorithm then tries to do a greedy reduction, *i.e.*, a replacement of an over-replicated content by c if possible. If not, it performs another switch, *i.e.*, by identifying some content under-replicated by at least 2, and creating a new replica in place of some perfectly replicated item, thereby augmenting the size of C_-. Hence, in at most $M - 1$ steps, the algorithm inflates the size of C_- to at least M, at which stage we know that a greedy reduction can be performed. This alteration between greedy reductions and switches is repeated until the size of C_- is at most $M - 1$, and each such content is missing *exactly one replica.*

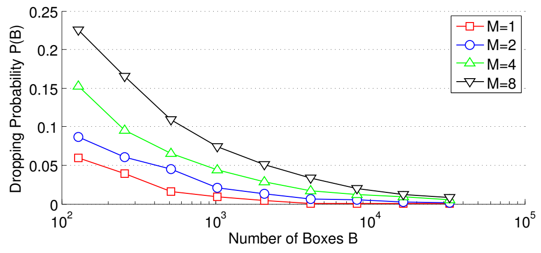


Figure 3: Dropping probability decreases fast with uniform slot strategy. Simulation in a single class with a catalog size of $C = 100$.

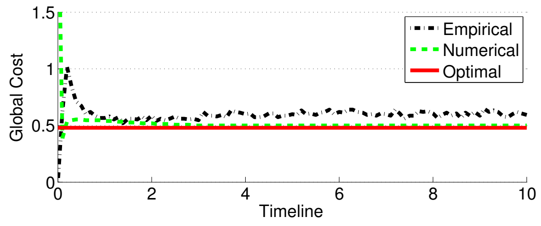


Figure 4: Decentralized optimization, content placement scheme and uniform-slot policy, under parameters $C = 1000, D = 10, \bar{B} = 1000, \bar{U} = 3, \bar{M} = 4$.

7. PERFORMANCE EVALUATION

In this section, we evaluate our algorithms under synthesized traces and a real-life BitTorrent traffic trace. We implement an event-driven simulator that captures box-level content placement and service assignment. In particular, we implement the solutions we proposed in Section 5 (decentralized optimization) and Section 6 (Designated Slot Placement). We also implement class trackers that execute the decentralized solution in Fig. 1.

In the rest of this section, all evaluations are performed using a cost matrix with random class-pairwise sampled uniformly from $[0, 1]$. The download cost from a box in the same class is 0, while the cost of downloading from the infrastructure is set to 3 for all classes.

7.1 Simulation on Synthesized Trace

First, we show that the uniform-slot policy achieves close-to-optimal service assignment, given that content-wise capacity constraints are respected. We focus on a single ISP and assign requests to individual boxes under the uniform-slot policy. Fig. 3 shows the loss probability under various settings of B and M. We utilize a synthesized trace with Poisson arrivals and exponentially distributed service time of mean one. Content popularity follows a Zipf distribution. We scale the total request rate to be proportional to the number of boxes. As predicted by the theory, the dropping probability quickly vanishes as B grows. For the same B, the dropping probability is higher when M is larger. When the storage is rich, our scheme tries to allocate popular content in caches in order to minimize cost. The locally absorbed requests are not counted in calculating the dropping probability. In fact, the effective requests come from unpopular contents, which is expected to generate a higher drop rate.

We next show the optimality of our full solution, again utilizing a synthesized trace generated as above. Content popularities are heterogeneous in different classes. Fig. 4 illustrates the average empirical cost per request, compared to (a) the fluid prediction under distributed optimization, when λ_c^d and $r_c^{dd'}$ are estimated perfectly, and (b) the optimal cost computed offline. Though the number of boxes is finite, and the above quantities are estimated empirically, the distributed algorithm converges close to the global optimum after a handful of rounds.

7.2 Simulation on BitTorrent Trace

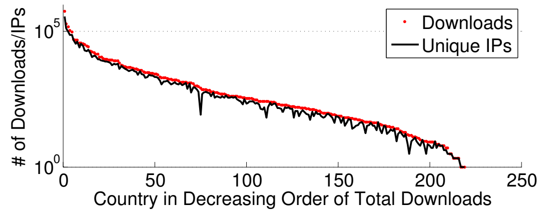
We next employ a real-life trace collected from the global Vuze network, one of the most popular BitTorrent clients. Our main motivation for using the BitTorrent trace is to validate our solution under realistic content popularity and access patterns.

Trace Collection and Evaluation Setup. Vuze clients issue a *put* each time they start downloading a file, and route the *put* to the 20 nodes whose IDs are closest to the file identifier. To collect these traces, we ran 1000 DHT nodes with randomly-chosen IDs, and logged all DHT *put* messages routed to our nodes. Therefore, our traces show all downloads for those files whose identifiers are close to the ID of one of our 1000 DHT nodes. Since the Vuze DHT has around 1 million nodes, and each file download is observed by 20 nodes, by running 1000 nodes we observe around $10^6/20/1000$, *i.e.*, 2% of all download requests. During 30 days we traced the downloads of around 2 million unique files by 8 million unique IP addresses. We determined the country of each IP using Maxmind's GeoLite City database [27]. We use the country-level geo-locality of BT users to organize them into classes. We do not model interclass cost, *e.g.*, latency, on geo-locality, as measuring and estimating such costs are outside the scope of this paper.

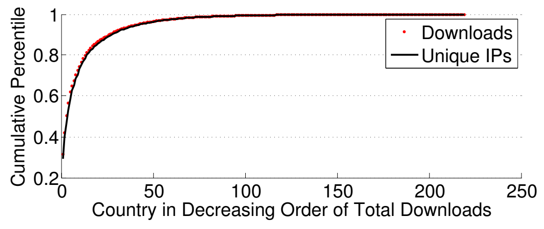
To limit the runtime of our event-driven simulations, we trim the traces by considering only the top 1,000 most popular files, which contribute 52% of the total downloads. Given a fixed number of boxes and cache size, there are very few copies of unpopular contents cached in these boxes. As a result, adding more contents will not significantly change the system configuration and the overall cost.

Fig. 5(a) illustrates the total number of download events and unique IPs, grouped by countries in a decreasing order of total downloads during the 30-day period, and the cumulative counts in Fig. 5(b). We select the top 20 countries as classes in our evaluation, comprising over 80% of all downloads. The trace shows that one user issues, on average, approximately one content request. We use 2 upload slots and 2 storage slots for each user (box) in our simulation.

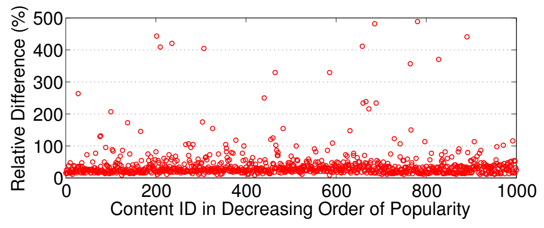
Previously we have shown the efficiency of our algorithm given fixed request rates. In practice, the content popularity may change over time. The content demand should remain sufficiently stable in order for our algorithm to work well and not oscillate wildly. In our evaluation, we measure content demands based on a 24-hour interval, and simply use the demand from the previous day to *project* the request rates for the next day. This undoubtedly only provides a conservative estimate for our solution, as a more accurate traffic prediction will further improve the performance of our approach. Fig. 5(c) shows the relative difference between the predicted rate and the true rate, grouped by content in decreasing order of popularity. Each data point is averaged over the



(a)



(b)



(c)

Figure 5: BitTorrent trace statistics. (a) Cumulative counts of downloads/boxes. (b) Per-country counts of downloads/boxes. (c) Predictability of content demand in 1-hour interval over 1 month.

entire 30-day period. The results show that demand is stable over 24 hrs for most content. High variations are due to new contents such as videos that reach a popularity peak during the first few days after their releases.

Evaluation results. We compare our solution to frequently used caching strategies, as well as a request routing heuristic. In particular, we implement Least Recently Used (LRU), and Least Frequently Used (LFU) caching algorithm. Each box implements the cache eviction policy locally. The recency and frequency counts apply to *all* contents, *i.e.*, including both local and remote requests. We also implement a local request routing heuristic, *i.e.*, the "closest" policy. When a download request is issued, all classes are examined in a greedy manner, *e.g.*, starting from its own class, to remote classes in an increasing order of cost. The request is accepted by a class whenever there exists at least one box in this class which stores the desired content and has one or multiple free slots. If no such a class is found, the request is directed to the CDN server. We implement the following solutions: (i) LRU-closest and (ii) LFU-closest, the combinations of caching and request routing heuristics, (iii) Decentralized, the full set of solution proposed in this paper, and (iv) Offline Optimal, which assumes the perfect knowledge of content demands and is the optimal solution to the global problem. The offline optimal provides a lower-bound for download costs, but does not reflect content shuffling costs between different periods.

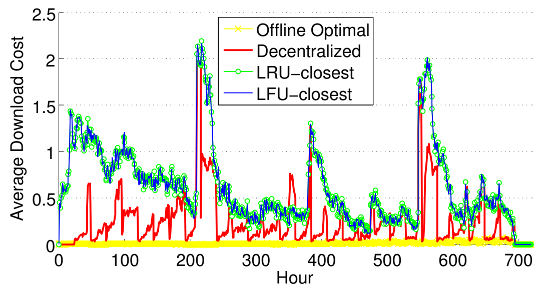


Figure 6: Performance of different algorithms over a real 30-day BitTorrent trace.

We evaluate the same BitTorrent trace under the four solutions. All scenarios begin with the same initial content placement over all boxes. Fig. 6 shows the average download cost (between 0 and 3) every hour for the entire 30-day period. Results demonstrate that our solution significantly reduces cross traffic between classes compared to the two heuristics. This is not surprising because both LRU and LFU are local heuristics that do not consider the global demand and optimize overall costs, and are implemented on top of a greedy local routing policy, which is certainly sub-optimal than a *jointly* optimized scheme. Our results provide an evidence that such an optimal gap can be quite large. The spiked cost is a consequence of content shuffles and temporary request drops when new popular contents are introduced. Our solution is able to quickly adapt to such changes and save a significant amount of costs.

8. SUMMARY

We offer a solution to regulate cross-traffic and minimize content delivery costs in decentralized CDNs. We present an optimal request routing scheme that can nicely accommodate user demands, an effective service mapping algorithm that is easy to implement within each operator, and an adaptive content caching algorithm with low operational costs. Through a live BitTorrent trace-based simulation, we demonstrate that our distributed algorithm is simultaneously scalable, accurate and responsive.

9. REFERENCES

[1] "Cisco visual networking index: Forecast and methodology, 2010-2015."

[2] "Nanodatacenters: http://www.nanodatacenters.eu/."

[3] "Boxee." http://www.boxee.tv/.

[4] C. Huang, A. Wang, J. Li, and K. W. Ross, "Understanding hybrid CDN-P2P: why Limelight needs its own Red Swoosh," in *NOSSDAV*, 2008.

[5] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez, "Greening the Internet with nano data centers," in *CoNEXT*, 2009.

[6] C. Huang, J. Li, and K. W. Ross, "Peer-assisted VoD: Making Internet video distribution cheap," in *IPTPS*, 2007.

[7] R. S. Peterson and E. G. Sirer, "Antfarm: Efficient content distribution with managed swarms," in *NSDI*, 2009.

[8] Y. Chen, Y. Huang, R. Jana, H. Jiang, M. Rabinovich, B. Wei, and Z. Xiao, "When is P2P technology beneficial for IPTV services," in *NOSSDAV*, 2007.

[9] Y. F. Chen, Y. Huang, R. Jana, H. Jiang, M. Rabinovich, J. Rahe, B. Wei, and Z. Xiao, "Towards capacity and profit optimization of video-on-demand services in a peer-assisted

IPTV platform," *Multimedia Systems*, vol. 15, no. 1, pp. 19–32, 2009.

[10] V. Misra, S. Ioannidis, A. Chaintreau, and L. Massoulié, "Incentivizing peer-assisted services: A fluid Shapley value approach," in *ACM SIGMETRICS*, 2010.

[11] "People's CDN: http://pcdn.info/."

[12] D. Han, D. G. Andersen, M. Kaminsky, K. Papagiannaki, and S. Seshan, "Hulu in the neighborhood," in *COMSNET*, 2011.

[13] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang, "Improving traffic locality in bittorrent via biased neighbor selection," in *ICDCS*, 2006.

[14] R. Cuevas, N. Laoutaris, X. Yang, G. Siganos, and P. Rodriguez, "Deep diving into BitTorrent locality," in *INFOCOM*, 2011.

[15] D. R. Choffnes and F. E. Bustamante, "Taming the torrent: a practical approach to reducing cross-ISP traffic in peer-to-peer systems," in *SIGCOMM*, 2008.

[16] V. Aggarwal, A. Feldmann, and C. Scheideler, "Can ISPs and P2P users cooperate for improved performance?," *ACM SIGCOMM Computer Communication Review*, vol. 37, pp. 29–40, July 2007.

[17] J. Wang, C. Huang, and J. Li, "On ISP-friendly rate allocation for peer-assisted VoD," in *Multimedia*, 2008.

[18] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz, "P4P: Provider Portal for (P2P) Applications," in *SIGCOMM*, 2008.

[19] I. Baev, R. Rajaraman, and C. Swamy, "Approximation algorithms for data placement in arbitrary networks," *SIAM Journal of Computing*, vol. 38, no. 4, 2008.

[20] S. C. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *INFOCOM*, 2010.

[21] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, "Placement algorithms for hierarchical cooperative caching," *Journal of Algorithms*, vol. 38, 2001.

[22] Y. Zhou, T. Z. J. Fu, and D. M. Chiu, "Modeling and analysis of P2P replication to support VoD service," in *INFOCOM*, 2011.

[23] W. Wu and J. C. S. Lui, "Exploring the optimal replication strategy in P2P-VoD systems: characterization and evaluation," in *INFOCOM*, 2011.

[24] B. R. Tan and L. Massoulié, "Optimal content placement for peer-to-peer video-on-demand systems," in *INFOCOM*, 2011.

[25] K. W. Ross, *Multiservice Loss Networks for Broadband Telecommunications Networks*. Springer-Verlag, 1995.

[26] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997.

[27] "http://www.maxmind.com/app/geolitecity."

[28] L. Massoulié, "Structural properties of proportional fairness," *The Annals of Applied Probability*, vol. 17, no. 3, 2007.

[29] H. Kushner and G. Yin, *Stochastic approximation and recursive algorithms and applications*. Springer, 2003.

[30] M. Benaïm and J.-Y. LeBoudec, "A class of mean field interaction models for computer and communication systems," *Perform. Eval*, pp. 11–12, 2008.

APPENDIX

A. PROOF OF THEOREM 1

After the conversion of (8d) and (8e) to equality constraints through (10), **GLOBAL** has the following properties. First, the objective (8a) is separable in the local variables $(\boldsymbol{r}^d, \boldsymbol{p}^d, y^d, \boldsymbol{z}^d)$, corresponding to each class. Second, the constraints (10a) and (10b) coupling the local variables are linear equalities. Finally, the remaining constraints (8b) and (8c) as well as the positivity constraints define a bounded convex domain for the local primal variables. These properties imply that the method of multipliers admits the distributed implementation in [26], Example 4.4., pp. 249–251, and the theorem follows. □

B. PROOF OF THEOREM 2

We partition the set of boxes $\mathcal{B}$ according to the contents they store in their cache. In particular, each of the boxes in $\mathcal{B}$ are grouped into L sub-classes $\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_L$, where all boxes in the i-th class $\mathcal{B}_i$, $1 \le i \le L$, store the same set of contents $\mathcal{F}_i \subset \mathcal{C}$ such that $|\mathcal{F}_i| = M$. We denote by $\mathcal{L} = \{1, 2, \dots, L\}$ and by $B_i = |\mathcal{B}_i|$ the number of boxes in the i-th sub-class. For $A \subseteq \mathcal{C}$, let $\mathsf{supp}(A) = \{i \in \mathcal{L} : \mathcal{F}_i \cap A \neq \emptyset\}$ be the set of sub-classes of boxes storing at least one content in A, and $\beta_i = \frac{B_i U}{BU}$, $i \in \mathcal{L}$ and $\rho_c = \frac{r_c}{BU}$, $c \in \mathcal{C}$. Note that, by our scaling assumption, when $B \to \infty$, the above quantities remain constant, and that (11) can be rewritten as: $\sum_{c \in A} \rho_c < \sum_{j \in \mathsf{supp}(A)} \beta_i$, $\forall A \subseteq \mathcal{C}$.

Let X_i be the number of empty slots in the i-th subclass. Then, under the uniform slot service assignment policy, the stochastic process $\mathbf{X} : \mathbb{R}_+ \to \mathbb{N}^L$ is a Markov process and can be described as follows:

$$X_i(t) = X_i(0) + E_i^+ \left(\int_0^t B_i U - X_i(\tau) d\tau \right)$$
$$-E_i^- \left(\int_0^t \sum_{c \in \mathcal{F}_i} r_c \frac{X_i(\tau)}{\sum_{j:c \in \mathcal{F}_j} X_j(\tau)} d\tau \right), \quad i \in \mathcal{L}, \tag{13}$$

where E_i^+, E_i^-, $i \in \mathcal{L}$, are independent unit-rate Poisson processes. Assume by convention that for all $i \in \mathcal{L}$ and all $c \in \mathcal{F}_i$, $\frac{X_i(\tau)}{\sum_{j:c\in\mathcal{F}_j} X_j(\tau)} = 0$ whenever $\sum_{j:c\in\mathcal{F}_j} X_j(\tau) = 0$. A mapping $\mathbf{x} : \mathbb{R}_+ \to [0,1]^L$ is a *fluid trajectory* of the system if it satisfies the following set of equations for all $i \in \mathcal{L}$.

$$x_i(t) = x_i(0) + \beta_i t - \int_0^t x_i(\tau) d\tau - \sum_{c \in F_i} \rho_c \int_0^t z_{c,i}(\mathbf{x}(\tau)) d\tau, \tag{14}$$

where $z_{c,i}$, $c \in \mathcal{F}_i$, are functions satisfying

$$z_{c,i}(\mathbf{x}) = \frac{x_i}{\sum_{j:c\in\mathcal{F}_j} x_j}, \text{ if } \sum_{j:c\in\mathcal{F}_j} x_j > 0, \text{ and} \tag{15a}$$

$$z_{c,i}(\mathbf{x}) > 0, \sum_{i \in \mathsf{supp}(\{c\})} z_{c,i} \le 1 \text{ otherwise} \tag{15b}$$

Given a vector $\mathbf{x}_0 \in [0,1]^L$, we define $S(\mathbf{x}_0)$ to be the set of fluid trajectories defined by integral equation (14) with initial condition $\mathbf{x}(0) = \mathbf{x}_0$.

LEMMA 3. *Consider a sequence of positive numbers* $\{B^k\}_{k\in\mathbb{N}}$ *such that* $\lim_{k\to\infty} B^k = +\infty$, *and a sequence of initial conditions* $\mathbf{X}^k(0) = [x_i^k]_{1\le i\le L}$ *s.t. the limit* $\lim_{k\to\infty} \frac{1}{B^k}\mathbf{X}^k(0) = \mathbf{x}_0$ *exists. Let* $\{\mathbf{X}^k(t)\}_{t\in\mathbb{R}_+}$ *denote the Markov process given by* (13) *given that* $B = B_k$, *and consider the rescaled process* $\mathbf{x}^k(t) = \frac{1}{B_k U}\mathbf{X}^k(t)$, $\quad t \in \mathbb{R}_+$. *Then for all* $T > 0$ *and all* $\epsilon > 0$, $\lim_{k\to\infty} \mathbf{P}\left(\inf_{\mathbf{x}\in S(\mathbf{x}_0)} \sup_{t\in[O,T]} |\mathbf{x}^k(t) - \mathbf{x}(t)| \ge \epsilon\right) = 0.$

PROOF. The proof is identical to the one in [28]. The only steps that we need to verify is that for every i and every $c \in F_i$, $X_i / \sum_{j:c\in\mathcal{F}_j} X_j$ is bounded (by 1), and at its points of discontinuity $\mathbf{x}'$ such that $\sum_{j:c\in F_j} X_j$ is zero, $\limsup_{\mathbf{x}\to\mathbf{x}'} X_i / \sum_{j:c\in\mathcal{F}_j} X_j = 1$. Both are easy to verify. □

Lemma 3 implies that (a) the set of fluid trajectories $S(\mathbf{x})$ is non-empty and (b) the rescaled process $\mathbf{x}^k$ converges on every finite set $[0,T]$ to a fluid trajectory, as $B \to \infty$, in probability. We therefore turn our attention to studying the asymptotic behavior of such fluid trajectories.

Given an $\mathbf{x}_0 \in [0,1]^L$, consider a fluid trajectory $\mathbf{x} \in S(\mathbf{x}_0)$. Since $z_{i,c}$ are bounded by 1, (14) implies that x is Lipschitz continuous (with parameter $\sum_c \rho_c$). By Rademacher's theorem, $\dot{x}$ exists almost everywhere and is given by

$$\dot{x}_i = \beta_i - x_i - \sum_{c\in F_i} \rho_c z_{c,i}(\mathbf{x}), \quad i \in \mathcal{L}. \tag{16}$$

Let $J = \lim_{t\to\infty} \bigcup_{\mathbf{y}\in[0,1]^L} \{\mathbf{x}(s), s \geq t : \mathbf{x}(0) = \mathbf{y}\}$ be the *limit set* [29] of ODE (16). Then, Lemma 3 implies (see Thm. 3 in Benaïm and Le Boudec [30]) that, as k tends to infinity, the support of the steady state probability of $\mathbf{X}^k$ converges to a subset of J. Thus, to show that the probability that queries for every content item succeed asymptotically almost surely, it suffices to show that $x_i^* > 0$ for every $\mathbf{x}^* \in J$. Indeed, let $I_0(\mathbf{x}) = \{i : x_i = 0\}$ be the zero-valued coordinates of $\mathbf{x}$ and $C(I) = \{c \in \mathcal{C} : \mathsf{supp}(\{c\}) \subseteq I\}$ denote the set of items stored only by classes in $I \subseteq \mathcal{L}$. Consider the following candidate Lyapunov function: $G(\mathbf{x}) = \sum_{i\in\mathcal{L}} \beta_i \log(x_i) - \sum_{i\in\mathcal{L}} x_i - \sum_{c\in\mathcal{C}} \rho_c \log\left(\sum_{j\in\mathsf{supp}(\{c\})} x_j\right)$, if $\mathbf{x} > 0$ and $G(\mathbf{x}) = -\infty$ otherwise.

Lemma 4. *Under* (11), *G is continuous in* $[0,1]^L$.

Proof. Consider a $\mathbf{x}'$ such that $I \equiv I_0(\mathbf{x}') \neq \emptyset$. Consider a sequence $\mathbf{x}^k \in [0,1]^L$, $k \in \mathbb{N}$, s.t. $\mathbf{x}^k \to \mathbf{x}$ in the $\|\cdot\|_\infty$ norm (or any equivalent norm in $\mathbb{R}^L$). We need to show that $\lim_{k\to\infty} G(\mathbf{x}^k) = -\infty$. If $I_0(\mathbf{x}^k) \neq \emptyset$ for some k, then $G(\mathbf{x}^k) = -\infty$; hence, w.l.o.g., we can assume that $\mathbf{x}^k \in (0,1]^L$. Then $G(\mathbf{x}^k) = A^k + B^k$, where $A^k = \sum_{i\in\mathcal{L}\setminus I} \beta_i \log(x_i^k) - \sum_{i\in\mathcal{L}} x_i^k - \sum_{c\in\mathcal{C}\setminus C(I)} \rho_c \log\big(\sum_{j\in\mathsf{supp}(\{c\})} x_j^k\big)$, and $B^k = \sum_{i\in I} \beta_i \log(x_i^k) - \sum_{c\in C(I)} \rho_c \log\left(\sum_{j\in\mathsf{supp}(\{c\})} x_j^k\right)$. The by the continuity of the log function, and the fact that $x_i' > 0$ for all $i \notin I$, it is easy to see that $\lim_{k\to\infty} A^k$ exists and is finite. On the other hand, if $C(I) = \emptyset$, B^k obviously converges to $-\infty$. Assume thus that $C(I)$ is non-empty. Partition I into classes $I_1, \ldots, I_m$ s.t. $x_i^k = y_\ell^k$ for all $i \in I_\ell$ (*i.e.*, all coordinates in a class assume the same value). Then

$$\begin{aligned}
B^k \leq& \sum_{i\in I} \beta_i \log(x_i^k) - \sum_{c\in C(I)} \rho_c \log\big(\max_{j\in\mathsf{supp}(\{c\})} x_j^k\big) \\
=& \sum_{\ell=1}^m \log(y_\ell^k) \sum_{i\in I_\ell} \beta_i - \sum_{\ell=1}^m \log(y_\ell^k) \sum_{c\in C(I)} \rho_c \mathbb{1}_{y_\ell^k = \max_{j\in\mathsf{supp}(\{c\})} x_j^k} \\
\leq& \sum_{\ell=1}^m \log(y_\ell^k) \big(\sum_{i\in I_\ell} \beta_i - \sum_{c\in C(I_\ell)} \rho_c\big)
\end{aligned}$$

since $\mathbb{1}_{y_\ell^k = \max_{j\in\mathsf{supp}(\{c\})} x_j^k} \leq \mathbb{1}_{\mathsf{supp}(\{c\})\cap I_\ell = \emptyset}$ and $\log(y_\ell^k) < 0$ for k large enough. Under (11), the above quantity tends to $-\infty$ as $k \to \infty$, and the lemma follows. ☐

Suppose that $I_0(\mathbf{x}(t)) = \emptyset$, *i.e.*. $\mathbf{x}(t) > 0$. Then (16) gives $\frac{d(\log x_i)}{dt} = \frac{\dot{x}_i}{x_i} = \frac{\beta_i}{x_i} - 1 - \sum_{c\in F_i} \rho_c \frac{1}{\sum_{j:c\in F_j} x_j} = \frac{\partial G}{\partial x_i}$ Hence, $\frac{dG(\vec{x}(t))}{dt} = \sum_i \frac{\partial G}{\partial x_i} \dot{x}_i = \sum_i x_i \left(\frac{\partial G}{\partial x_i}\right)^2 \geq 0$, *i.e.*, when at $\mathbf{x}$, G is increasing as time progresses under the dynamics (16). This, implies that if $\mathbf{x}(t) > 0$ then the fluid trajectory will stay bounded away from any $\mathbf{x}'$ s.t. $I_0(\mathbf{x}) \neq \emptyset$, as $G(\mathbf{x}') = -\infty$ and by Lemma 4 to reach such an x' the quantity $G(\mathbf{x}(t))$ would have to decrease, a contradiction.

Suppose now that $I = I_0(\mathbf{x}(t)) \neq \emptyset$. We will show that $I_0(\mathbf{x}(t+\delta)) = \emptyset$, for small enough δ. Our previous analysis for the case $I_0(\mathbf{x})$ therefore applies and the theorem, as the limit set L cannot include points $\mathbf{x}^*$such that $I_0(\mathbf{x}^0) \neq \emptyset$. By (14) and (15), fluid trajectories are Lipschitz continuous; hence, for δ small enough, $x_i(t+\delta) > 0$ for all $i \notin I$. By (16), $\sum_{i\in I} \frac{dx_i}{dt} = \sum_{i\in I} \beta_i - \sum_{\substack{i\in I\\ c\in F_i}} \rho_c z_{c,i}(\mathbf{x}) \overset{(15a)}{=} \sum_{i\in I} \beta_i - \sum_{\substack{i\in I\\ c\in F_i\cap C(I)}} \rho_c z_{c,i}(\mathbf{x}) \overset{(15b)}{\geq} \sum_{i\in I} \beta_i - \sum_{c\in C(I)} \rho_c \overset{(11)}{>} 0$. Hence, for δ small enough, there exists at least one $i \in I$ such that $x_i(t+\delta) > 0$. Given that x_i, $i \notin I$, will stay bounded away from zero within this interval, this implies that within δ time (where δ small enough) all coordinates in I will become positive. Hence, $I_0(\mathbf{x}(t+\delta)) = \emptyset$, for small enough δ. ☐

C. PROOF OF LEMMA 2

We provide a constructive proof below, calculating q_c^d that satisfy (12). If $M^d = 1$, the lemma trivially holds for $q_c^d = p_c^d$. Now suppose that $M^d \geq 2$. Let $\epsilon = 1 - \sum_{c\in\mathcal{C}} r_c^{\cdot d}/B^d U^d$ and $\epsilon_c = p_c^d - r_c^{\cdot d}/B^d U^d$. From (8d) and (8e), we have that $\epsilon > 0$ and $\epsilon_c > 0$. Sort ϵ_c in an increasing fashion, so that $\epsilon_{c_1} \leq \epsilon_{c_2} \leq \ldots \leq \epsilon_{c_{|\mathcal{C}|}}$. If $\epsilon_{c_1} \geq \epsilon$, then the lemma holds for $q_c^d = r_c^{\cdot d}/B^d U^d + \epsilon/|\mathcal{C}|$. Assume thus that $\epsilon_{c_1} < \epsilon$. Let $k = \max\{j : \sum_{i=1}^j \epsilon_{c_i} < \epsilon\}$. Then $1 \leq k < |\mathcal{C}|$, as $\sum_{c\in\mathcal{C}} \epsilon_c \overset{(8b)}{=} M - 1 + \epsilon > M - 1 > \epsilon$ for $M \geq 2$. Then, $\epsilon' = \epsilon - \sum_{i=1}^k \epsilon_{c_i} > 0$, by the definition of k. Let $q_{c_i}^d = r_c^{\cdot d}/BU + \epsilon_{c_i}$ for $i \leq k$ and $q_{c_i}^d = r_c^{\cdot d}/BU + \epsilon'/(|\mathcal{C}| - k)$ for $i > k$. Then $q_{c_i}^d = p_{c_i}^d > \lambda_c/BU$ for $i \leq k$. For $i > k$, $q_{c_i}^d > r_c^{\cdot d}/BU$ as $\epsilon' > 0$ while $\epsilon'/(|\mathcal{C}| - k) \leq \epsilon' \leq \epsilon_{c_{k+1}}$, as otherwise $\sum_{i=1}^{k+1} \epsilon_{c_i} < 1$, a contradiction. Moreover, $\epsilon_{c_{k+1}} \leq \epsilon_{c_i}$ for all $i \geq k$, so $q_{c_i}^d \leq r_{c_i}^{\cdot d}/BU + \epsilon_{c_i} \leq p_c^d$. Finally, $\sum_c q_c^d = \sum_{i=1}^k (r_{c_i}^{\cdot d}/BU + \epsilon_{c_i}) + \sum_{i=k+1}^{\mathcal{C}} r_{c_i}^{\cdot d}/BU + \epsilon' = \sum_c r_c^{\cdot d}/BU + \epsilon = 1$, and the lemma follows. ☐

D. PROOF OF THM. 3

Every modification in the first phase of the algorithm incurs at most one cache write, and reduces the l_1 norm—*i.e.*, the imbalance—between the vectors $B\boldsymbol{q}$ and $B\boldsymbol{q'}$ by 2. As such, the first phase terminates in at most $B\alpha/2$ operations, at a total cost of at most $B\alpha/2$ writes. In the second phase, every greedy reduction reduces the l_1 distance—*i.e.*, the imbalance—between vectors $B\boldsymbol{\pi}$ and $B\boldsymbol{\pi'}$ by 2, at the cost of one write operation. In the third phase, every switch maintains the imbalance constant; moreover, there can be no more than $M-1$ consecutive switch operations that must immediately be followed by a greedy reduction. As such, the imbalance reduces by 1 in at most M write operations. At termination of the third phase, since C_- is a has size at most $M-1$, and each item is missing at most one replica, the imbalance is at most $2M$. Since after phase one the imbalance was $N\sum_c |\pi_c - \pi_c'| \leq B(\alpha + \beta)$, the total number of write operations in the second and third phase is at most $(M-1)(B/2)(\alpha + \beta)$. ☐