# Optimizing Cost and Performance for Content Multihoming

Hongqiang Harry Liu[⋆] Ye Wang[⋆] Yang Richard Yang[⋆×] Hao Wang[†] Chen Tian[⋆]
[⋆]Yale University [†]Google [×]SplendorStream
{hongqiang.liu, ye.wang, yang.r.yang, tian.chen}@yale.edu wanghao@google.com

## ABSTRACT

Many large content publishers use multiple content distribution networks to deliver their content, and many commercial systems have become available to help a broader set of content publishers to benefit from using multiple distribution networks, which we refer to as *content multihoming*. In this paper, we conduct the first systematic study on optimizing content multihoming, by introducing novel algorithms to optimize both performance and cost for content multihoming. In particular, we design a novel, efficient algorithm to compute assignments of content objects to content distribution networks for content publishers, considering both cost and performance. We also design a novel, lightweight client adaptation algorithm executing at individual content viewers to achieve scalable, fine-grained, fast online adaptation to optimize the quality of experience (QoE) for individual viewers. We prove the optimality of our optimization algorithms and conduct systematic, extensive evaluations, using real charging data, content viewer demands, and performance data, to demonstrate the effectiveness of our algorithms. We show that our content multihoming algorithms reduce publishing cost by up to 40%. Our client algorithm executing in browsers reduces viewer QoE degradation by 51%.

**Categories and Subject Descriptors:** C.2.3 [**Computer Communication Networks**]: Network Operations.

**Keywords:** Content Delivery, Multiple CDNs, Optimization.

## 1. INTRODUCTION

Many content publishers on the Internet use multiple content distribution networks (CDNs) to distribute and cache their digital content. We refer to content publishing using multiple content distribution networks as *content multihoming*. In our recent survey, we found that all major content publishers such as Netflix, Hulu, Microsoft, Apple, Facebook, and MSNBC use content multihoming.

Content publishers adopt content multihoming to aggregate the diversity of individual CDN providers on features, performance and commitment [7]. For example, one CDN may provide good coverage for locations 1 and 2, whereas another CDN provides good coverage for locations 2 and 3. To deliver content to viewers from all three locations, a content publisher may need to use both CDNs.

Given the wide usage and potential benefits of content multihoming, many commercial systems supporting content multihoming have recently been deployed (*e.g.*, [9, 11, 16, 17, 21, 27]), so that more content publishers can benefit from content multihoming. However, these commercial products either use ad hoc approaches or do not provide details on their designs. No previous studies on how to effectively utilize content multihoming are known.

In this paper, we attempt to provide a framework and a set of novel algorithms to optimize the benefits of content multihoming. We ask a simple question: Given that content multihoming allows a content object to be delivered from multiple CDNs, which CDN(s) should a content publisher use to deliver each object to each content viewer requesting this object, so that the publisher optimizes its benefits from content multihoming? This question is the key to efficiently utilizing content multihoming, since its solutions can be implemented directly with the flexible request routing mechanisms (*e.g.*, DNS CNAME, HTTP Redirect from servers, and client scripts in end hosts) in modern content delivery infrastructures.

An answer to this simple question, however, is not immediately obvious. Consider the current common approach of choosing, for each content viewer, the best performing CDN among all candidate CDNs. This approach, despite its simplicity, has multiple issues. First, although the chosen CDN may provide the highest level of performance, for example, satisfying that 99% viewers do not see quality of experience (QoE) degradation, the cost of the chosen CDN can be much higher than another CDN with a slightly lower, but still high enough level of 95%. Second, there are often multiple CDNs with comparable and sufficient levels of performance at a given region, *e.g.*, in US. One common approach to break ties in such cases is to pick the CDN with the lowest cost. However, the costs of CDNs, in particular, pay-as-you-go CDNs such as Amazon CloudFront, are volume based and non-linear. The cost of one object assignment depends on the other assignments. Third, there are locations where even the best performing CDN falls short. For example, a content publisher may have a QoE target of 95%, but the best performing CDN at some location achieves only 90%.

In this paper, we answer the preceding question by designing two algorithms: (1) an efficient optimization algorithm executing at content publisher to compute content distribution guidance, and (2) a simple algorithm executing at individual content viewers to follow the guidance with local adaptation. Either algorithm can be deployed alone, but together they benefit the most.

Specifically, the publisher optimization algorithm, named CMO, computes CDN assignments considering many real factors: nonlinear, multi-region CDN traffic charging, per-request charging, content licensing restrictions, CDN feature availability, and CDN performance variations. The CMO algorithm is novel and highly efficient. For example, when considering traffic cost only, the complexity of CMO is polynomial and independent of the number of content objects, whereas the complexity of simple enumeration is exponential in the number of content objects.

The local viewer algorithm provides a capability for a content viewer to make efficient usage of multiple servers from multiple CDNs, with a preference ordering on the usage of CDN edge servers provided by the content publisher. Inspired by TCP AIMD and using a simple prioritized assignment mechanism, the algorithm adapts the usage of multiple CDNs, achieving a performance level that no single CDN/server can achieve alone.

We implement both of the algorithms and conduct systematic, extensive evaluations using real charging data, content viewer demands, and CDN performance to demonstrate the effectiveness of our algorithms. We show that our content multihoming algorithms reduce publishing cost by up to 40%. Our implementation of the client algorithm running in Adobe Flash enabled browsers reduces viewer QoE degradation by 51%.

*SIGCOMM'12,* August 13–17, 2012, Helsinki, Finland.
Copyright 2012 ACM 978-1-4503-1419-0/12/08 ...$15.00.

## 2. RELATED WORK

The importance of content multihoming has led to substantial industrial related work. We divide the industrial efforts into three categories [7]. The first category is software systems which we name CDN switchers (*e.g.*, [1, 21]) and integrators (*e.g.*, [19, 27]. For example, the One Pica Image CDN extension [21] of the Magento Commerce platform provides an API to support the integration with multiple CDNs, including Amazon S3, Coral CDN, Mosso/Rackspace Cloud Files, and any CDN server or service that supports FTP, FTPS, or SFTP. Their objective, however, is not on the algorithms to effectively use multiple CDNs, but rather on usability issues such as seamless switching from one CDN provider to another. Commercial systems such as [19, 27] provide CDN services based on aggregation of multiple CDNs. They can benefit from using our algorithms.

Going beyond the CDN switchers and integrators is a category of systems named CDN Load balancers. There are many CDN load balancers commercially available, including Cotendo CDN balancer, LimeLight traffic load balancer, Level 3 intelligent traffic management, and Dyn CDN manager. Some of these systems offer quite flexible rules to split CDN traffic among multiple CDNs. For example, Cotendo CDN balancer supports rules with weighted allocation, geographic location, geographic distance, time of day, and any combination. In particular, the weighted allocation rule allows a publisher to specify: $x$ percent to CDN one, $y$ percent to CDN two, and so on. A key missing component of these existing systems, however, is the key algorithms to compute the allocation (*e.g.*, the percentages). Hence, the output of our optimizer can be used to configure these systems.

There are also client based CDN load balancers. One interesting example is Conviva [9], whose video player plug-in performs continuous video quality monitoring, and could perform automatic CDN and/or source server switching during video playback. The exact details of their algorithm, however, are unknown.

A third category of related industrial efforts is CDN interconnect. In [20], a CDN interconnect (CDNi) architecture has been proposed so that a content publisher contracts with a few upstream CDNs, who may delegate some requests to downstream CDNs. The delegation relationship can be recursive to form a directional delegation graph, and all of the involved CDNs are said to form a CDN federation [5]. Our algorithms can be extended to the CDNi setting by considering a set of connected CDNs as a single logical CDN.

So far content multihoming has not been a focus of academic research. A related recent academic work is a measurement study of Netflix [2]. The paper shows that similar to many content publishers, Netflix uses content multihoming. The paper conducts a measurement study and shows that there are indeed potential performance benefits of using content multihoming.

There has been some recent studies on intra-CDN optimizations which can be used in conjunction with content multihoming. The authors of [3, 22, 23, 24] study how CDNs can utilize information on load of edge servers, network conditions, and locations/bandwidth of clients to improve CDN request routing. Our work complements these studies by optimizing the assignment of contents into CDNs.

We refer to our system as content multihoming to draw an analogy with traditional Internet multihoming (*e.g.*, [13]). However, content multihoming is quite different from traditional ISP multihoming. For example, while ISPs typically have a uniform price based on traffic, CDNs charge customers by regions.

## 3. BACKGROUND AND NOTATIONS

We start by introducing the background and notations. Table 1 provides a reference for the major notations used in this paper.

There are three key types of entities being managed in content multihoming: (1) content objects; (2) viewers of contents; (3) distribution networks that cache contents from origin networks to serve content viewers.

**Content object**: A content publisher can have a large number of content objects such as videos and images. Let $N$ denote the total number of content objects. An object has many properties. In the context of content distribution, the performance requirement, the size, and the popularity of an object are its key properties [4]. Let $s_i$ be the size of object $i$. We introduce object popularity when we next introduce content viewers.

| | |
|---|---|
| $N$ | Number of content objects. |
| $K$ | Number of CDNs. |
| $A$ | Set of fine-grained location areas. |
| $s_i$ | Size of object $i$. |
| $n_i^a$ | Number of requests for object $i$ from area $a$. |
| $i^a$ | Location object: object $i$ requested from location area $a$. |
| $\alpha_k^r$ | Set of location areas served by charging region $r$ of CDN $k$. |
| $t_k^r$ | Charging volume of CDN $k$ at its charging region $r$. |
| $C_k^r()$ | Charging function of CDN $k$ for its charging region $r$. |
| $F_k$ | Set of location objects that CDN $k$ can serve. |
| $p_{i,k}^a$ | CDN $k$'s performance for object $i$ at location area $a$:<br>fraction of times CDN $k$ can deliver $i^a$ with sufficient QoE. |
| $x_{i,k}^a$ | CDN guidance: fraction of $n_i^a$ requests directed to CDN $k$. |
| $\psi_k$ | CDN assignment: set of location objects assigned to CDN $k$. |

**Table 1: Summary of key notations.**

**Content viewer (client)**: There can be a large number of content viewers requesting content objects. These content viewers can be distributed across multiple geographical areas. The specific geographical areas depend on the particular requirements of a content publisher. For generality and conceptual clarity, let $A$ be the set of all geographical areas, say all cities or countries. Note that in this challenging general case the size of $A$ can be large, on the order of thousands. Let $a \in A$ denote a location area.

The popularity of an object among content viewers is location dependent [12]. Let $n_i^a$ denote the number of times that object $i$ will be requested, during a time interval (say a month), from content viewers located at location area $a$.

We also use $n_i^a$ to encode *licensing restrictions* that a content publisher often needs to enforce in practice. Specifically, if content viewers from a location area $a$ should not receive a content object $i$, $n_i^a$ should be 0.

**Content distribution network (CDN)**: A key reason of content multihoming is to aggregate the *capability-geography expertise* of different CDNs, as different CDNs can have quite different performance and cost characteristics, at different geographic regions. On the other hand, as we will see, such differences are a major source of intrinsic complexity when optimizing content multihoming. In this paper, we assume that the set of CDNs is given. Let $K$ be the number of CDNs, and we use $k$ and $j$ to index individual CDNs.

First consider *performance*. Figure 1 shows the edge server footprints of three real CDNs: Amazon CloudFront, MaxCDN and ChinaCache. When a content viewer from a location area $a$ requests a content object through CDN $k$, a well designed CDN will choose an edge server (or several servers) that is close to $a$ to serve the request, since a short latency from edge servers to end users is typically needed to achieve good content delivery performance [15]. Comparing the geographical footprints of the three CDNs shown in Figure 1, one can anticipate that CloudFront and MaxCDN are more likely to provide better performance in US and Europe, while ChinaCache may perform well in China. None of the three covers regions such as Russia and Africa.

To quantify the performance, we conduct performance measurements using 600 PlanetLab nodes at different locations to request content objects from three CDNs (CloudFront, MaxCDN, and Liquid Web). Since performance metrics are dependent on the content

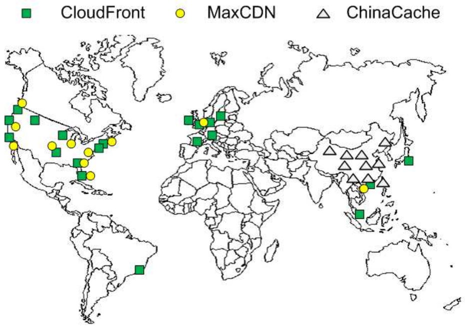


**Figure 1: Edge server distributions of three CDNs.**

type, and streaming media is a major content type [8], we evaluate the delivery of streaming media. Table 2 shows the measured success rates (fraction of clients with no video freeze) of the CDNs to deliver streaming objects encoded at three different streaming rates (1 Mbps, 2 Mbps, and 3 Mbps) to some representative locations. For example, the entry for Liquid Web/Spain shows the success rates when PlanetLab viewers from Spain request from Liquid Web: if the object is encoded at 1 Mbps, 99.4% of the viewers can receive at the encoding rate; for a 2 Mbps object, only 47.3% of the viewers can receive at the encoding rate; for a 3 Mbps object, almost no viewers can receive at the encoding rate. The measurement results show clearly that the usability of a CDN depends on both the object (*e.g.*, a video encoded at 1 Mbps or higher) and the location of the viewer. We refer to object $i$ being requested by viewers at a specific location area $a$ as a *location object*, denoted as location object $i^a$.

| | CloudFront | | | MaxCDN | | | Liquid Web | | |
|---|---|---|---|---|---|---|---|---|---|
| US | 99.9 | 99.9 | 99.9 | 99.2 | 98.4 | 97.8 | 99.3 | 96.1 | 92.1 |
| Brazil | 100 | 100 | 99.9 | 98.6 | 70.5 | 24.4 | 99.6 | 0 | 0 |
| Austria | 99.9 | 99.9 | 99.8 | 97.6 | 96.7 | 95.3 | 97.0 | 42.2 | 0 |
| Spain | 99.9 | 99.9 | 99.9 | 98.7 | 96.6 | 95.1 | 99.4 | 47.3 | 0.2 |
| Japan | 99.9 | 99.9 | 99.9 | 97.5 | 95.8 | 77.1 | 99.7 | 0 | 0 |
| China | 99.9 | 99.9 | 99.8 | 91.1 | 24.7 | 0 | 1.6 | 0 | 0 |
| Australia | 100 | 100 | 99.9 | 94.7 | 89.5 | 0 | 99.7 | 0 | 0 |

**Table 2: Measured CDN performance $p^a_{i,k}$ (3 content objects at streaming rates 1/2/3 Mbps).**

To precisely characterize the performance of CDN $k$, in this paper, we define $p^a_{i,k}$ as the fraction of times (*e.g.*, 90%) that CDN $k$ can deliver $i^a$ at the encoding rate of the object $i$. One may define $p^a_{i,k}$ for other contexts (*e.g.*, for images) and use other metrics (*e.g.*, 95-percentile latency). Practically $p^a_{i,k}$ can be obtained from measurements or service level agreements (SLA) of CDNs.

Next consider *costs*. Different regions may have different resource (*e.g.*, bandwidth, electricity) costs. Different CDNs may operate at different scales at different regions to negotiate different volume discounts. Hence, different CDNs' prices might be various, and one CDN may charge differently at each region.

Figure 2 shows the real charging structures of two CDNs: Amazon CloudFront and MaxCDN. We show these two structures because they are public and represent typical CDN charging structures. We make three observations. First, each CDN groups the locations of its edge servers into multiple regions and each region may have a different pricing model. We refer to each such region as a *charging region*. For example, CloudFront divides into 5 charging regions: US, EU, South America (SA), Japan (JP), and Singapore/Hong Kong (SHK). MaxCDN divides into 2 charging regions: US/EU/SA, and Asia Pacific (AP). The total charge of a CDN to a content publisher is the sum of the charges at all of the charging regions. Second, denote the total traffic originated from the edge servers of a CDN located at a charging region during a billing period as the *charging volume* of the charging region; then the charging function of each charging region is a nonlinear concave function of the charging volume. Third, there can be large price diversity within a CDN as well as across CDNs. For example, CloudFront's charge for South America for "next 100 TB" is \$0.18/GB, which is 3 times that for US at the same traffic volume.

To precisely express the charging of CDNs, we let $\alpha^r_k$ be the set of location areas served by charging region $r$ of CDN $k$. For viewers from a location area $a$, each CDN has its own strategy to select servers in some $r$ to serve them. This strategy is controlled by CDN $k$ but can be observed by content publishers [25]. For instance, in our measurements, requests from Beijing are redirected to CloudFront's JP region but to MaxCDN's US/EU/SA region. Let $t^r_k$ denote the charging volume of CDN $k$ at its charging region $r$, during a billing period. Specifically, the value of $t^r_k$ is computed as the total traffic delivered during the billing period to content viewers at location areas who are assigned to $\alpha^r_k$ by CDN $k$. Then the total charge of CDN $k$ to the content publisher is a sum of the charges at individual charging regions. Let $C^r_k()$ denote the charging function of CDN $k$ for its charging region $r$.

## 4. CONTROL FRAMEWORK

We adopt a general, practical content publishing control framework shown in Figure 3. A central *Optimizer* computes configurations to direct viewer requests to CDNs. The configurations are sent to a DNS system, HTTP redirector, or a manifest-file server system to implement direction for specific viewer requests.

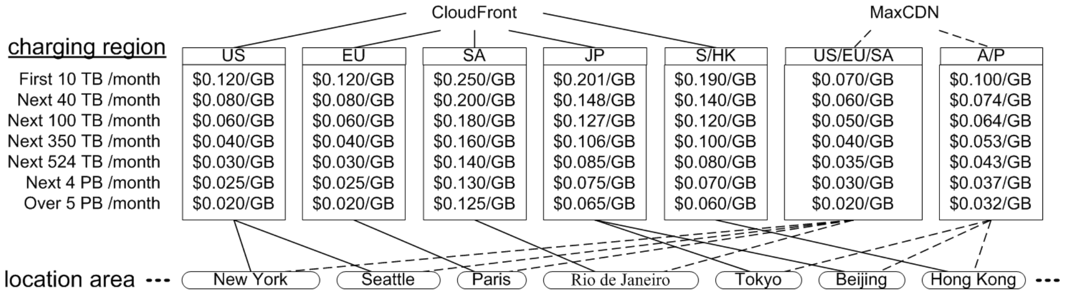
In particular, we distinguish two types of clients according to their capabilities. This distinction affects our problem formulation. The first type is passive clients. A key characteristic of passive clients is that they use one CDN edge server at a time. Although multiple CDNs and/or multiple servers from one CDN are available in content multihoming, such traditional clients at content viewers use only a single CDN server to serve a particular content object request [2]. For such clients, we assume that the results of content multihoming optimization are implemented only by server side mechanisms such as DNS redirection.

Specifically, suppose that content publisher `cp.com` uses DNS-based CDN redirection. For simplicity, say the publisher assigns content object $i$ URL `http://obj-i.cp.com`. When a client requests this link, a DNS query from the client or the local DNS server of the client to resolve `obj-i.cp.com` can be sent to the DNS server of the content publisher, labeled `CPDNS` in Figure 3. By looking up the IP address of the client or the local DNS resolver of the client, `CPDNS` obtains the location area $a$. Using the output from the Optimizer, `CPDNS` returns to the client the CNAME of a chosen CDN $k$. As an example shown in Figure 3, CloudFront is chosen. Note in real implementations, as we will see in Section 6, URL assignment is not necessarily per-object. Alternative implementations (*e.g.*, using HTTP-based CDN redirection) are similar.

The second type is active clients. Such clients include an adaptation algorithm (*e.g.*, in Adobe Flash ActionScript) to utilize multiple CDN servers when retrieving a single content object. In particular, when the service rate of one CDN server is insufficient, an active client can use additional servers (from the same CDN or backup CDNs) to make up the deficit. The additional CDNs/servers are provided to the adaptation algorithm through a manifest file from the content publisher. Such manifest files are already used by some clients such as the Netflix clients. As shown in Figure 3, the CNAMEs of two CDNs are returned to an active client.

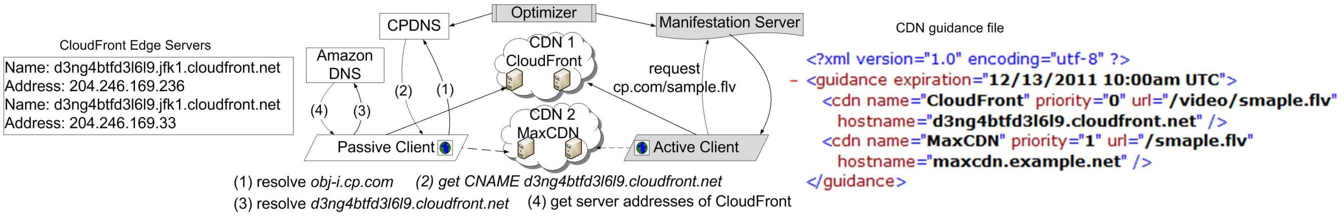
## 5. PROBLEM STATEMENTS

With the preceding background and control framework, our problems are easy to state. Note that there is much flexibility in the deployment of our control framework. There can be settings with only passive clients, or only active clients, or a hybrid. We first state the



| charging region | US | EU | SA | JP | S/HK | US/EU/SA | A/P |
|---|---|---|---|---|---|---|---|
| First 10 TB /month | $0.120/GB | $0.120/GB | $0.250/GB | $0.201/GB | $0.190/GB | $0.070/GB | $0.100/GB |
| Next 40 TB /month | $0.080/GB | $0.080/GB | $0.200/GB | $0.148/GB | $0.140/GB | $0.060/GB | $0.074/GB |
| Next 100 TB /month | $0.060/GB | $0.060/GB | $0.180/GB | $0.127/GB | $0.120/GB | $0.050/GB | $0.064/GB |
| Next 350 TB /month | $0.040/GB | $0.040/GB | $0.160/GB | $0.106/GB | $0.100/GB | $0.040/GB | $0.053/GB |
| Next 524 TB /month | $0.030/GB | $0.030/GB | $0.140/GB | $0.085/GB | $0.080/GB | $0.035/GB | $0.043/GB |
| Next 4 PB /month | $0.025/GB | $0.025/GB | $0.130/GB | $0.075/GB | $0.070/GB | $0.030/GB | $0.037/GB |
| Over 5 PB /month | $0.020/GB | $0.020/GB | $0.125/GB | $0.065/GB | $0.060/GB | $0.020/GB | $0.032/GB |

**Figure 2: Charging structures of CloudFront and MaxCDN.**



Name: d3ng4btfd3l6l9.jfk1.cloudfront.net
Address: 204.246.169.236
Name: d3ng4btfd3l6l9.jfk1.cloudfront.net
Address: 204.246.169.33

```
  <?xml version="1.0" encoding="utf-8" ?>
- <guidance expiration="12/13/2011 10:00am UTC">
    <cdn name="CloudFront" priority="0" url="/video/smaple.flv"
      hostname="d3ng4btfd3l6l9.cloudfront.net" />
    <cdn name="MaxCDN" priority="1" url="/smaple.flv"
      hostname="maxcdn.example.net" />
  </guidance>
```

(1) resolve *obj-i.cp.com* *(2) get CNAME d3ng4btfd3l6l9.cloudfront.net*
(3) resolve *d3ng4btfd3l6l9.cloudfront.net* (4) get server addresses of CloudFront

**Figure 3: Content multihoming control framework (shaded components include our contributions).**

problem in the passive client only setting. An extension to active client only setting is straightforward and follows. Combining them together is also straightforward and hence omitted.

## 5.1 Passive Client

Since a generic problem formulation has benefits, we define our problem using generic constrained programming. Specifically, we formulate the content multihoming optimization problem as consisting of two objectives:

**QoE guarantee**: First and foremost, for every object $i$ and location area $a$, if $n_i^a > 0$, content multihoming should assign one or more CDNs for viewers from location area $a$ requesting object $i$ to achieve a QoE target. Each assigned CDN $k$ should satisfy two requirements: (1) CDN $k$ is providing the required features (*e.g.*, streaming vs download) to deliver object $i$; (2) $p_{i,k}^a$ exceeds the performance target ($\bar{p}$). Define $F_k$={$i^a$: CDN $k$ can provide the features to deliver object $i$ and $p_{i,k}^a \geq \bar{p}$.}. In other words, $F_k$ is the set of location objects that CDN $k$ can serve. Note that it is $F_k$ that allows us to distinguish between streaming vs downloading content, as it depends on object QoE metrics and features (*e.g.*, HTTP-streaming).

**Cost optimization**: Under the QoE guarantee constraint, content multihoming may balance the load to multiple CDNs, in particular to minimize the total cost. Let $x_{i,k}^a$ denote the fraction of the $n_i^a$ requests that is directed to CDN $k$. Hence, each $x_{i,k}^a$ is an optimization variable with a valid value range between 0 and 1. We state the problem **Q** as:

$$\begin{aligned} \underset{\{x_{i,k}^a\}}{\text{minimize}} \quad & C(\{x_{i,k}^a\}) = \sum_k \sum_{\alpha_k^r} C_k^r(\sum_{a \in \alpha_k^r} \sum_i x_{i,k}^a n_i^a s_i) \\ \text{subject to} \quad & \forall i, a, n_i^a > 0 : \sum_k x_{i,k}^a = 1, x_{i,k}^a \geq 0; \\ & \forall k, i, a, i^a \notin F_k : x_{i,k}^a = 0. \end{aligned}$$

The first constraint states that each demand for an object $i$ at a location $a$ should be served. The second constraint states the QoE constraint. In other words, if CDN $k$ cannot provide sufficient performance or feature for a location object $i^a$, no content viewers from location area $a$ for object $i$ should be directed to CDN $k$. Note that the QoE constraint may lead to infeasibility. Since feasibility can be checked efficiently, we assume feasibility.

One can add additional constraints (*e.g.*, CDN capacity constraints) and consider additional cost factors (*e.g.*, storage, per request). We will first focus on the basic constraints and cost factors. In Section 6.2, we will discuss some extensions. Note that we start with a basic formulation based on longer-term statistics/prediction to select CDNs. In Section 7, we will add active clients for online adaptation to form a more complete 2-level optimization framework. Better traffic predictions (or publisher-knowledge) on $n_i^a$ can be input plugins to the problem formulation.

After computing a solution (*i.e.*, $\{x_{i,k}^a\}$), the Optimizer sends the solution to `CPDNS` in the control framework of Section 4 to implement it. Typically, the computation should be performed each billing cycle (*e.g.*, a month) but can be more frequently.

## 5.2 Active Client

An active client allows the usage of multiple CDNs to serve the same request. One might think that this will add substantial complexity to the preceding formulation. But as we will see, it is a simple extension of the preceding problem definition.

Without loss of generality, consider that each active client uses two CDNs: one primary and one backup. Consider the following set of CDNs: each CDN is a "virtual CDN" that consists of a pair of CDNs, say $k^{'} = (k, j)$ where $k$ is the primary CDN and $j$ is the backup. Then we will have a similar problem formulation as the preceding formulation in problem **Q**.

First consider the QoE Guarantee. Define $F_{k'}$={$i^a$: both CDN $k$ and CDN $j$ can provide the features to deliver object $i$ and $p_{i,k}^a \cup p_{i,j}^a \geq \bar{p}$.}, where $\cup$ denotes the joint reliability of the two CDNs.

Next consider the objective function. Assume that each primary CDN $k$ still delivers the same amount of traffic. The backup CDN $j$ incurs an additional traffic $1 - p_{i,k}^a$ fraction of the time. One may verify that $C(\{x_{i,k'}^a\})$ and $C(\{x_{i,k}^a\})$ have similar structure and can be solved with the same method.

After computing a solution(*i.e.*, $\{x_{i,k'}^a\}$), where each $k^{'}$ is a pair of CDNs, the Optimizer sends the solution to manifest file servers to return two CDNs for each active client request.

# 6. COMPUTING OPTIMIZATION

We now develop techniques to solve the problem defined in the preceding section. Since the problems for the passive clients and active clients have the same format, we use the passive client formulation: Problem **Q**. One might consider solving **Q** using standard LP, but this can be intractable: 500K objects, 200 locations, and 3 CDNs will imply 300M variables, 100M constraints, and to minimize a *concave* function. No generic solver we know can handle this case, so we develop our specific algorithm.

Our strategy is to first transform the problem to a combinatorial assignment problem in Section 6.1. Then, in Section 6.2 we develop a novel, efficient algorithm that computes an optimal assignment without enumerating all of the exponentially many possibilities. We discuss extensions in Section 6.3.

## 6.1 Location Object Assignment

At a first glance of the problem **Q**, one might think about using convex programming (*e.g.*, [6]) to solve the problem. Unfortunately, the objective function $C(\{x^a_{i,k}\})$, which we target to minimize, is a *concave* function. Hence, traditional, efficient convex programming does not apply.

On the other hand, the concavity of the objective does lead to one observation: there is a minimizer of problem **Q** such that each location object is put into only one CDN. Precisely, we have:

LEMMA 1. *There exists a minimizer of $C(\{x^a_{i,k}\})$ for problem* **Q***, in which for each location object $i^a$, there exists a CDN $k^*$ such that $i^a \in F_{k^*}$, and $x^a_{i,k^*}$ = 1.*

Consider one such solution, and let $\psi$ denote the assignment (or mapping), according to the solution, from each location object $i^a$ to its assigned single CDN $k$: $\psi(i^a) = k$. Let $\psi_k$ denote the set of location objects that are assigned to CDN $k$. If $\forall k, \forall i^a \in \psi_k$ we have $i^a \in F_k$, we call $\psi$ a *feasible assignment*.

The above interpretation of the solution allows us to change problem **Q** into an assignment-based formulation, as shown in Figure 4. For each location object, the figure shows the candidate CDNs that the location object can be assigned to. CDN $k$ is a candidate for location object $i^a$ only if $i^a \in F_k$. The problem then is to assign each location object to a single CDN to minimize the cost.

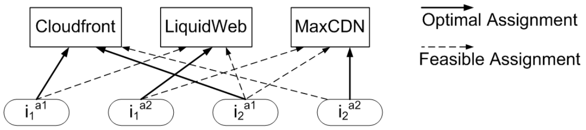


**Figure 4: Q can be formulated as an assignment problem.**

An advantage of the discrete assignment formulation is that it allows enumeration. A straightforward approach to finding an optimal assignment is to enumerate all possibilities, and select the best one among the feasible assignments. We know that each location object $i^a$ can be assigned to one of the $K$ CDNs. Hence, the total number of assignments is $K^{|A|N}$. For $K = 2$ or 3, $|A|$ on the order of thousands and $N$ hundreds of thousands, direct enumeration is practically infeasible. In other words, the assignment formulation allows enumeration, but naive enumeration does not work.

## 6.2 Efficient and Optimal Assignment

Our key insight to substantially reduce the complexity is that the naive enumeration of all of the exponential number of assignments is unnecessary. Instead, we need to consider only a polynomial number of assignments.

**Basic idea:** Specifically, consider the space of all possible assignments illustrated by the space on the left of Figure 5(a), where each assignment is shown as a point. A feasible assignment is shown as an white box while an infeasible one is shown as a black box. Naive enumeration evaluates every assignment, ignores infeasible assignments, and picks a feasible one that gives the best outcome.

Now instead of looking at the space of assignments, we look at the space of the *outcomes* of the assignments, illustrated by the right space in Figure 5(a). Each assignment point in the left space has a corresponding outcome point in the right outcome space. Specifically, the outcome of an assignment $\psi$ is a vector, with each element of the vector representing the charging volume at charging region of CDN. Let $V_\psi$ denote the multi-dimensional charging volume vector representing the outcome of an assignment $\psi$. We will develop the exact representation shortly.

Since the objective function of our problem **Q** is a concave function of the charging-volume vector, we know from concave optimization theory that we need to evaluate the objective function only over the extremal points of the convex hull of the charging volume vectors produced by feasible assignments. In other words, if the charging volume vector $V_\psi$ resulted from a feasible assignment $\psi$ is not an extremal point, the vector can be expressed as a convex combination of those resulted from some other feasible assignments, and hence there is no need to evaluate $\psi$. Figure 5(a) illustrates that we need to consider those $V_\psi$ marked with an "x". As we will see, the number of extremal points is polynomial and below we develop our efficient algorithm to identify them.

**Representing each location object as a vector**: The foundation of our basic idea is based on considering the resulting charging volumes of an assignment $\psi$ as a vector $V_\psi$. We now introduce a representation of each location object $v = i^a$ as a vector to allow easy aggregation on the outcome of an assignment. This representation is quite simple but involves some notation complexity at the beginning. The benefit of the representation is that it provides essential insight and simplification during our development.

We first introduce charging region intersections. Recall that each CDN $k$ defines a mapping from a location area $a$ to one of its serving charging regions $\alpha^1_k$, $\cdots$, $\alpha^{R_k}_k$, where $R_k$ is the number of charging regions of CDN $k$. Note that $\alpha^1_k, \cdots, \alpha^{R_k}_k$ provides a partition of all location regions $A$. An intrinsic complexity of multiple CDNs is the heterogeneity of their charging regions. Define the "intersections" of the charging regions of the $K$ CDNs. Let $\alpha^{r_1 r_2 \cdots r_K}$ denote the intersection of the charging regions $r_1$ of CDN 1, $r_2$ of CDN 2, and $r_K$ of CDN K. Then a total of $R = R_1 * R_2 \cdots R_K$ intersections are defined. Figure 6(a) illustrates a setting of two CDNs with 3 and 2 charging regions respectively. At most $R = 6$ non-empty intersections may be defined.

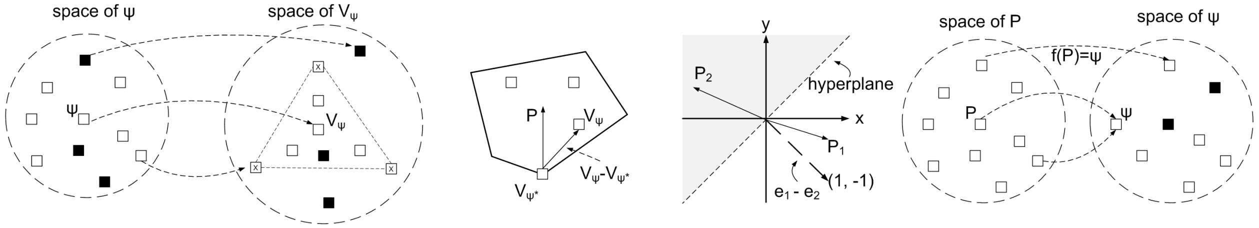
With charging region intersections, we can represent each location object as a vector. Specifically, given the set of charging region intersections, one can observe that each location area $a$ belongs to one and only one of the charging region intersections. Fix one ordering of the intersections. Then we can convert the traffic of each location object $v = i^a$ to an $R$-dimension vector with all elements except one being 0. The position of the non-zero element is the intersection that the location area $a$ belongs to, and the value at the position is $n^a_i s_i$. When it is clear from the context, we use $v$ to either represent the name of a location object $i^a$ or the vector. Figure 6(b) shows the vector representations of two location objects.

By representing each location object $v = i^a$ as a $R$ dimensional vector, we introduce a simple, linear outer-production operator to reflect the effect of assigning $v$ to CDN $k$. Let $e_k$ be a unit $K$-dimensional vector whose only non-zero element is at the $k$-th position and the value at the k-th position is 1. Define $v \otimes e_k$, which reflects the effect of assigning $v$ to CDN $k$, as producing a $R \times K$ matrix such that $v$ is at the k-th column and the other columns are zero. Figure 6(c) shows four examples, when we assign two location objects to two CDNs. For example, the first example shows $v_1 \otimes e_1$; that is, assigning $v_1$ to CDN 1.

Given this definition of the outer-product and an assignment $\psi$, we can calculate the outcome of $\psi$. Recalling that $\psi_k$ is the set of location objects assigned to CDN $k$, we have:

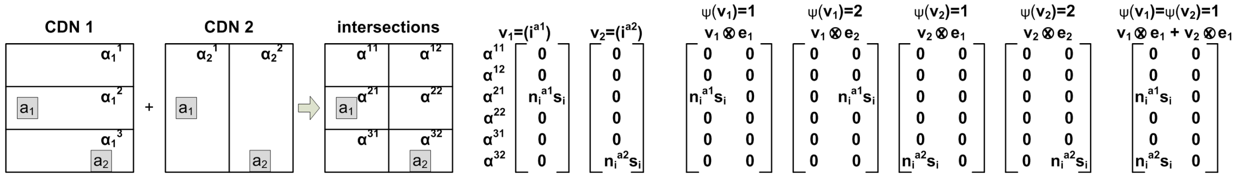
$$V_\psi = [\sum_{v \in \psi_1} v, \ldots, \sum_{v \in \psi_K} v] = \sum_v v \otimes e_{\psi(v)} \in \mathbb{R}^{R \times K}. \qquad (1)$$

We now go back to identifying extremal assignments.



(a) mapping from ψ to $V_\psi$ (b) P and $V_\psi$-$V_{\psi^*}$ have a positive inner product (c) $P_1$ and $P_2$ are interior points (d) solution strategy

**Figure 5: An example illustrating the basic idea to solve problem Q.**



(a) charging region intersections (b) location-object as a vector (c) location-object can choose either CDN 1 or CDN 2 (d) both select CDN 1

**Figure 6: An example illustrating the charging-intersections.**

**Extremal assignments v.s. separation vector**: The basis of our technique to identify the extremal points of a set of points is through the fundamental Separation Lemma [14]. Specifically, define $V_\Psi = \{V_\psi\}$, where $\psi$ is a feasible assignment. Then a sufficient and necessary condition for a specific $V_{\psi^*} \in V_\Psi$ to be an extremal point of $V_\Psi$ is that there exists a vector $P$ in the same dimension space whose inner product $\langle P, V_\psi - V_{\psi^*}\rangle > 0$ for any other $V_\psi \in V_\Psi$. In other words, the inner product of $P$ and $V_{\psi^*}$ is smaller than the inner product between $P$ and any other point $V_\psi \in V_\Psi$. For instance, in Figure 5(b) , $\langle P, V_\psi - V_{\psi^*}\rangle > 0, \forall \psi \neq \psi^*$.

Hence, a strategy to identify the set of all extremal points is to compute a set of $P$s and from each $P$ we compute an extremal feasible assignment. Denote the computation from $P$ to an assignment as $f(P)$. Figure 5(d) illustrates a set of $P$s and a mapping function $f()$. We want the following conditions on the set of $P$s and $f()$ : (1) *computationally efficient*: both the set of $P$s and $f(P)$ are easy to compute; (2) *valid*: each $f(P)$ is an extremal feasible assignment; (3) *exhaustive*: for each extremal feasible assignment $\psi^*$, there exists one $P$ in the set of $P$s, such that $f(P) = \psi^*$. Below, we show a set of $P$s and a function $f(P)$ satisfying the preceding conditions.

**Function** $f(P)$: We start by developing $f(P)$. We have the following CDN Identification Lemma.

LEMMA 2 (CDN IDENTIFICATION LEMMA). *A feasible assignment $\psi^*$ is extremal among the set of all feasible assignments if and only if $\exists P \in \mathbb{R}^{R\times K}$ such that $\forall v, k$, $k \neq \psi^*(v) \wedge v \in F_k$: $\langle P, v \otimes e_{\psi^*(v)}\rangle < \langle P, v \otimes e_k\rangle$.*

PROOF. The right to the left is straightforward, as the right condition is stronger than the Separation Lemma condition, when one applies Expression (1). We prove the left to right. Given the Separation Lemma, we know that $\exists P \in \mathbb{R}^{R\times K}$ such that $\langle P, V_{\psi^*}\rangle < \langle P, V_\psi\rangle$, for any feasible $\psi \neq \psi^*$. Consider any $v$ and $k$ such that $v \in F_k$ and $k \neq \psi^*(v)$, let $\psi'$ be the assignment obtained from $\psi^*$ by moving $v$ to CDN $k$; that is, $\psi'(v) = k$ and $\psi'(u) = \psi^*(u)$, $\forall u \neq v$. Since $\psi'$ is also a feasible assignment, it follows that $\langle P, V_{\psi^*}\rangle < \langle P, V_{\psi'}\rangle$, hence $\langle P, v \otimes e_{\psi^*(v)}\rangle - \langle P, v \otimes e_k\rangle = \langle P, V_{\psi^*}\rangle - \langle P, V_{\psi'}\rangle < 0$. □

We name the lemma CDN Identification Lemma because it is the foundation to develop $f(P)$. Given a $P$ in the lemma, we can compute a corresponding extremal assignment $\psi$ efficiently: for each location object $v = i^a$, iterate among all feasible CDNs $k$ for the object (*i.e.,* $i^a \in F_k$), we compute $\langle P, v \otimes e_k\rangle$. We assign $v$ to the (unique) CDN $k$ attaining the minimal value: $\psi(v) = k$.

**Set of** $P$**s**: We consider the following set of $P$s: a $P$ satisfies that all of the elements in $\{\langle P, v \otimes e_k\rangle | \forall k : v \in F_k\}$ are distinct:

$$\{P : \forall k, j, v, k \neq j \wedge v \in F_k \cap F_j : \langle P, v \otimes (e_k - e_j)\rangle \neq 0\}. \quad (2)$$

Since the conditions are stronger than those from the CDN Identification Lemma, we know that each such $P$ can compute an extremal assignment.

Geometrically, the condition that $P$ satisfies $\langle P, v \otimes (e_k - e_j)\rangle \neq 0$ is equivalent to that $P$ is not on the hyperplane that is orthogonal to $v \otimes (e_k - e_j)$. Denote this hyperplane as $[v \otimes (e_k - e_j)]^\perp$ and let $\mathbb{H}$ be the set of all these hyperplanes. Hence, a $P$ satisfying all conditions in (2) is not on any of the hyperplanes in $\mathbb{H}$. In other words, $P$ should be an interior point in a cell created with hyperplanes in $\mathbb{H}$ as boundaries. Efficient algorithms (*e.g.,* [26]) exist to enumerate one interior point from each cell.

**Exhaustiveness**: From the preceding development, it should be clear that we have developed a set of $P$s and the function $f$ which is computationally efficient, and $f(P)$ is valid. The only remaining issue is whether we satisfy the exhaustive requirement by enumerating an arbitrary interior $P$ from each cell. First, we have

PROPOSITION 1. *If $\psi^*$ is an extremal feasible assignment, $\exists P$ making $f(P) = \psi^*$ and $P$ is an interior point of a cell in $\mathbb{H}$.*

PROOF. Suppose we find an extremal assignment $\psi^*$ from a point $Q$ that is not an interior point of any cell in $\mathbb{H}$. Our method is to construct another point $Q'$, from $Q$, that is indeed an interior point of some cell in $\mathbb{H}$ and that $f(Q') = \psi^*$.

Since $Q$ is not interior, $\exists v_0, k_0 \neq j_0$ such that $\langle Q, v_0 \otimes (e_{k_0} - e_{j_0})\rangle = 0$. Consider $P(\epsilon) = Q + \epsilon \cdot v_0 \otimes (e_{k_0} - e_{j_0})$ where $\epsilon \in \mathbb{R}^1$. Therefore $\forall v, k \neq j$, $\langle P(\epsilon), v \otimes (e_k - e_j)\rangle = \langle Q, v \otimes (e_k - e_j)\rangle + \epsilon \cdot \langle v_0 \otimes (e_{k_0} - e_{j_0}), v \otimes (e_k - e_j)\rangle$ is a continuous function of $\epsilon$. When $|\epsilon'| > 0$ is small enough: (1) For $\forall v \neq v_0, k \neq k_0$ or $j \neq j_0$, $\langle P(\epsilon'), v \otimes (e_k - e_j)\rangle \neq 0$ and has the same sign ('+' or '-') with $\langle Q, v \otimes (e_k - e_j)\rangle$; (2)$\langle P(\epsilon'), v_0 \otimes (e_{k_0} - e_{j_0})\rangle = \epsilon' \cdot \|v_0 \otimes (e_{k_0} - e_{j_0})\|^2 \neq 0$.

This means that $P(\epsilon')$ is on one less hyperplanes in $\mathbb{H}$ than $Q$. Moreover, since $\psi^*$ is extremal, we have $\langle Q, v \otimes (e_{\psi^*(v)} - e_k)\rangle < 0$ $\forall v, k, k \neq \psi^*(v) \wedge v \in F_k$, so it follows from (1) that $\langle P(\epsilon'), v \otimes (e_{\psi^*(v)} - e_k)\rangle < 0$ as well, hence $f(P(\epsilon')) = \psi^*$ by Lemma 2. This process can be repeated to yield a $Q'$ that is not on any hyperplanes in $\mathbb{H}$ and that $f(Q') = \psi^*$. □

A potential issue is that the interior $P$ from the preceding lemma may not be the one that our algorithm uses. However, we have the following result, and establish the exhaustiveness of our approach.

LEMMA 3. *Interior points from the same cell find the same extremal feasible assignment.*

PROOF. Let $P_1$ and $P_2$ are two interior points from the same cell. Suppose their corresponding extremal assignments $\psi_1^* \neq \psi_2^*$, then $\exists v_0$ which has $\psi_1^*(v_0) \neq \psi_2^*(v_0)$. According to Lemma 2:

$$\langle P_1, v_0 \otimes (e_{\psi_1^*(v_0)} - e_{\psi_2^*(v_0)})\rangle < 0$$
$$\langle P_2, v_0 \otimes (e_{\psi_1^*(v_0)} - e_{\psi_2^*(v_0)})\rangle > 0$$

which contradicts with that $P_1$ and $P_2$ are from the same cell. □

**Redundancy elimination**: On the surface, we need to enumerate all cells created by a total of $|A|NK(K-1)$ hyperplanes, where $|A|N$ is due to the number of possibilities for $v$, which is the number of location objects, and $K(K-1)$ is due to the number of possibilities for $(e_k - e_j)$. However, some of the $|A|NK(K-1)$ hyperplanes are redundant. Specifically, if one vector is just the scaling of another vector, they define the same hyperplane. For example, $v \otimes (e_k - e_j)$ and $v \otimes (e_j - e_k)$ give the same hyperplane. Hence, we need to consider only distinct pairs of $k$ and $j$. Also, consider two location objects $v_1 = i_1^{a_1}$ and $v_2 = i_2^{a_2}$. If their vector representations satisfy $v_1 = \lambda v_2$, where $\lambda$ is a scalar, then they define the same hyperplane, for each pair of $k$ and $j$. In other words, all location objects mapped to the same charging region intersection define the same hyperplane for each pair of $k$ and $j$. Hence, the number of unique hyperplanes is at most $R\binom{K}{2}$, which is independent of the number of content objects.

**Algorithm 1**: CMO(V, $\{F_k\}$)

```
   Input: V: location objects to be assigned.
   Input: {F_k}: K CDNs and their feasibility sets.
   Output: optAs: optimal assignment
1  /* Step 1: Identify hyperplanes */
2  HPs ← ∅ ;
3  foreach $v \in V$ do
4      vVec = vAsVector(v) ;
5      foreach distinct $(k, j)$ pairs do
6          if ($v \in F_k \wedge v \in F_j$) then
7              hpCandidate = $normalize([vVec \otimes (e_k - e_j)]^{\perp})$ ;
8              if ($hpCandidate \notin HPs$) then
9                  HPs += hpCandidate
10 /* Step 2: Compute interior points from hyperplanes */
11 Ps ← computePs(HPs);
12 /* Step 3: Evaluate extremal assignments identified by Ps */
13 optAs ← null;
14 foreach $P \in Ps$ do
15     /* compute extremal assignments $\psi$ identified by P */
16     $\psi$ ← null ;
17     foreach $v \in V$ do
18         optOuter ← $+\infty$;
19         foreach CDN $k$ do
20             if $v \in F_k \wedge \langle P, v \otimes e_k\rangle < optOuter$ then
21                 $\psi(v) \leftarrow k$ ;
22                 optOuter ← $\langle P, v \otimes e_k\rangle$
23     /* compare new extremal $\psi$ with current optAs */
24     if ($\psi > optAs$) then
25         optAs ← $\psi$
26 return optAs;
```

**Algorithm**: For completeness, we specify the content multihoming optimization (CMO) algorithm in Algorithm 1. There are many ways implementing computePs(), and we choose [26] as it can be easily parallelized for multiple CPU cores and computers.

**Example**: To help readers understand the CMO algorithm, we apply it to the simplest setting of two CDNs and both use one global charging region. This is a setting where one can solve problem **Q** using intuition. Specifically, in this setting, we can divide the location objects into 3 categories: $V_1$: the location objects that can be assigned to only CDN 1; $V_2$: the location objects that can be assigned to only CDN 2; and $V_3$: the location objects that can be assigned to either CDN 1 and CDN 2. Then the only remaining issue is to determine the assignments of objects in $V_3$. One can verify that the correct strategy is that we compare the objective function values of two alternatives: (1) assigning all objects in $V_3$ to CDN 1, with objects in $V_1$ and $V_2$ preassigned to their respective CDNs; and (2) assigning all objects in $V_3$ to CDN 2, with objects in $V_1$ and $V_2$ preassigned to their respective CDNs.

Now, we see how CMO works. In Step 1 (Lines 2 to 9), the algorithm computes that there is only one hyperplane defined by [1, -1]$^{\perp}$. In Step 2, ComputePs computes two interior Ps $P_1$ and $P_2$, where $P_1$ is a vector in the lower right half-space (x-coordinate is larger than the y-coordinate) of Figure 5(c), and $P_2$ is one in the upper left half-space (x-coordinate is smaller than the y-coordinate). In Step 3, the algorithm first evaluates $P$=$P_1$ to compute an extremal assignment. For each location object $v$=$i^a$, if it is in $V_1$ or $V_2$, the algorithm assigns it to the only feasible CDN. If $v \in V_3$, the value of $\langle P_1, v \otimes e_1\rangle$ is the x-coordinate of $P_1$ times the traffic volume of object $v$, and the value of $\langle P_1, v \otimes e_2\rangle$ is the y-coordinate of $P_1$ times the traffic volume of $v$. Since $P_1$ is chosen in Step 2 such that the x-coordinate is larger than the y-coordinate, $P_1$ produces the extremal assignment of assigning all objects in $V_3$ to CDN 2. The algorithm next evaluates $P$=$P_2$ and produces the extremal assignment of assigning all objects in $V_3$ to CDN 1. At Line 24, the algorithm compares the two cases and picks the better one. Hence, it produces the intuitive result. For general settings that we can no longer appeal to intuition, the algorithm computes the optimal assignment efficiently.

## 6.3 Extensions

The CMO algorithm developed in the preceding section applies to concave charging functions or charging functions that can be converted to or approximated by concave charging functions. In this section, we discuss extensions to handle practical issues on CDN subscription levels, per-request costs and dynamic streaming.

**CDN subscription levels:** For some CDNs, a content publisher must subscribe to a usage level (*e.g.*, maximum traffic volume) and pay a fix fee to the subscription level. To handle such a charging model, we treat each subscription level as an individual CDN with a capacity constraint. Note that the basic CMO algorithm needs to be slightly extended to handle capacity constraints [18].

**Per-request cost:** Besides charging for traffic, some CDNs also include charges for the number of requests. For instance, CloudFront charges $0.0075 per 10,000 HTTP requests in US. Consider that CloudFront charges $0.12/GB for the first 10 TB traffic as shown in Figure 2. One can calculate that if the object sizes are less than 6.25 KB, then the per-request charge can be higher than the traffic charge. Hence, the per-request charge can be the major cost for content publishers providing small objects (*e.g.*, small images).

Extending Algorithm 1 to consider both traffic and per-request charge is relatively straightforward. Specifically, in the preceding section, each location object is represented as a $R$-dimension vector with one non-zero element at the charging region of the object. An extension to include per-request charge is to represent each location object as a $R+1$ dimension vector, with the one added dimension representing the number of requests for the object.

| CDN | Traffic Charge | Per-request Charge |
|---|---|---|
| CDN 1 | $1.0 per GB | $0.0 per request |
| CDN 2 | $0.1 per GB | $0.1 per request |

| Object | Size | #Request | Traffic | Vector Representation |
|---|---|---|---|---|
| $v_1$ | 0.01 GB | 30 | 0.3 GB | [0.30 GB, 30] |
| $v_2$ | 0.01 GB | 49 | 0.49 GB | [0.49 GB, 49] |
| $v_3$ | 0.025 GB | 20 | 0.5 GB | [0.50 GB, 20] |
| $v_4$ | 1.0 GB | 1 | 1.0 GB | [1.00 GB, 1] |

**Table 3: Example of CMO with per-request cost extension.**

We demonstrate the extension using an example setting (Table 3): 4 objects, 2 CDNs with one charging region (*i.e.,* $R = 1$):

- First the 4 objects are represented as 4 2D vectors (see Table 3).
- Lines 3 to 9 construct four corresponding hyperplanes: $h_1$ = [0.3,-0.3,30,-30]$^\perp$, $h_2$ = [0.49,-0.49,49,-49]$^\perp$, $h_3$ = [0.5,-0.5,20,-20]$^\perp$, $h_4$ = [1,-1,1,-1]$^\perp$. After normalization and de-duplication (Lines 7, 8), only three are left: [1,-1,100,-100]$^\perp$, [1,-1,40,-40]$^\perp$, [1,-1,1,-1]$^\perp$; *i.e.,* first two are redundant and only 1 is needed.
- Line 11 finds 6 interior points: $P_1$ = [1,0,1,0], $P_2$ = [-1,0,-1,0], $P_3$ = [70,0,-1,0], $P_4$ = [-70,0,1,0] , $P_5$ = [20,0,-1,0], $P_6$ = [-20,0,1,0].
- Lines 14 to 22 find 6 extremal object assignments ($\psi$ :={CDN1 objects}{CDN2 objects}): $\psi_1$ = {}{$v_1$, $v_2$, $v_3$, $v_4$}, $\psi_2$ = {$v_1$, $v_2$, $v_3$, $v_4$}{}, $\psi_3$ = {$v_1$, $v_2$} {$v_3$, $v_4$}, $\psi_4$ = {$v_3$, $v_4$} {$v_1$, $v_2$}, $\psi_5$ = {$v_1$, $v_2$, $v_3$} {$v_4$}, $\psi_6$ = {$v_4$} {$v_1$, $v_2$, $v_3$}.
- Lines 23 to 25 enumerate the costs of the 6 extremal object assignments, and identify optimal assignment $\psi_5$. As a comparison, simple enumerations need to consider 16 assignment possibilities (4 objects each with 2 possible assignments).

**Multiple streaming rates:** A content publisher can encode the same video object at multiple rates, and a client can switch among the encoding rates dynamically online. To extend CMO for this setting, one can consider each video at each encoding rate as an independent object and then derive the number of requests to each such object. Specifically, suppose a total of $n_i^a$ requests for video $i$ from location area $a$ when there is no multi-rate. Consider two encoding rates, say 1x and 2x. Then according to client access bandwidth distribution at location area $a$ and the publisher dynamic streaming algorithm, one can derive $n_{i_1}^a$ and $n_{i_2}^a$ for the two encoding rates from $n_i^a$ [18].

# 7. ACTIVE CLIENTS

An active client is provided with a list of CDNs to use when requesting a single content object. The list may come from the result of our optimization algorithm in the preceding section or another source of guidance. Even though our optimization algorithm considers performance constraints, the filtering is based on long-term statistics. Hence, the objective of the active client is to adapt to specific real-time CDN performance dynamics, in particular, to improve QoE during CDN server failures.

## 7.1 Adaptation Problem Statement

An active client receives guidance in the format of a list of CDNs where each CDN has a priority value. Without loss of generality, we consider two CDNs: the first *primary* CDN, and the second *backup* CDN. For example, the example in Figure 3 gives an active client two CDNs with priority 0 (primary) and priority 1 (backup) respectively. We assume that each individual CDN on the list provides a small number (say, 1 or 2) preferred edge servers through its request routing mechanism (*e.g.,* DNS resolution or HTTP redirect). Let $H$ denote the set of all candidate edge servers. As an example, if the primary CDN provides two servers ($h_{11}$ and $h_{12}$) and the backup CDN provides one server ($h_{21}$), then $H = \{h_{11}, h_{12}, h_{21}\}$. Each server in $H$ has a priority value, which is the priority value of its CDN. At time $t$, an active client maintains a subset $H_{active}$ of $H$ under the following guidelines:

- *QoE protection (feasibility)*: the combined available bandwidth of the servers in $H_{active}$ can provide the target QoE;
- *Prioritized guidance*: the available bandwidth of a higher priority server should be used before that of a lower priority server;
- *Low session overhead*: the overhead of redistributing load among same-priority servers is unnecessary unless it reduces the number of servers used concurrently to reduce the number of connections overhead.

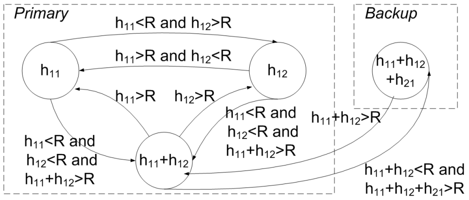


**Figure 7: An active-client control diagram.**

Based on the preceding guidelines, we derive a simple control state transition diagram as the control objective of an active client. Figure 7 shows an example control diagram where each node represents the current subset of edge servers that the active client uses. The client starts in the state $\{h_{11}\}$, indicating that the client starts from downloading from $h_{11}$.

Note that one can introduce other transitions using our control state diagram based approach.

## 7.2 Adaptation Algorithm

We implement the control diagram using the classic window-based AIMD (Additive Increase Multiplicative Decrease) scheme, based on a key observation that there is an analogy between the traditional congestion control and active client adaptation. In particular, if we consider the flow from each server to the client as a link, then we are solving a rate allocation problem among the links, where two essential mechanisms are needed: (1) the rate on a link should be reduced if the link is overloaded; and (2) a probing scheme is needed to utilize newly available capacity.

Specifically, for each sever $h$, the client maintains a *request window* size $w_h$ (KB), which is the current upper bound on the outstanding request load from the client to $h$ per $T$ seconds, where $T$ is a configuration parameter. For simplicity, we refer to each KByte in the request window as a *piece*. We use the classic AIMD algorithm as a base to adjust the window sizes: if $h$ and the network from $h$ to the client has enough capacity to finish the requested pieces, $w_h$ is linearly increased; otherwise, $w_h$ is multiplicatively decreased. Our client uses HTTP range request to query a set of pieces in one HTTP request message. After the range request is finished or a time out happens after $T$ seconds, an `onResponse` event handler is triggered. We implement the AIMD algorithm in `onResponse` listed as Algorithm 2.

**Algorithm 2**: onResponse(h, reqPieceSet$_h$, rcvdPieceSet$_h$)

```
if rcvdPieceSet_h = reqPieceSet_h and $w_h$ = reqPieceSet_h.size() then
    $w_h \leftarrow w_h + 1$;
if rcvdPieceSet_h $\subset$ reqPieceSet_h then
    $w_h \leftarrow \max(1, w_h/2)$;
/* Update piece status */
foreach piece i $\in$ reqPieceSet_h do
    if piece i $\in$ rcvdPieceSet_h then
        // extract piece i's data, add to video buffer;
        pieceStatusMap[i] $\leftarrow$ DOWNLOADED;
    else
        pieceStatusMap[i] $\leftarrow$ NOT_ASSIGNED;
avail_h $\leftarrow$ true; /* mark $h$ avaiable for new assignment */
updateAssignment();
```

However, our problem is also different from the traditional congestion control problem, and hence we need to introduce two novel and interesting techniques.

**Total workload control**: Naive usage of traditional AIMD will imply that all servers can be fully utilized to achieve a download rate that is as high as possible. However, this can be unnecessary for the content viewer. Specifically, to achieve QoE of streaming content, the client only needs to sustain a sufficient downloading rate (*e.g.,* the video encoding rate $\omega$ KB/s).

Based on this observation, we apply *total workload control* to appropriately limit the usage of the CDN servers. In our design, every $T$ seconds, a client calls `releaseLoad()`, shown in Algorithm 3, to release $\omega \cdot T$ new pieces to be downloaded. Note the client may download more than $\omega \cdot T$ pieces in $T$ seconds if there are incomplete pieces in previous downloading periods.

**Algorithm 3**: releaseLoad()

```
/* reqLimit: the "newest" piece to be assigned for downloading */
reqLimit ← reqLimit + $\omega \cdot T$;
// set pieceStatusMap of new pieces NOT_ASSIGNED
updateAssignment();
```

**Prioritized assignment**: Total workload control does not yet achieve our adaptation goal. In particular, the load may still spread unnecessarily to too many servers. For example, a single primary server $h_{11}$ has enough capacity to serve the client, but a backup server $h_{21}$ may be also used unnecessarily for downloading, if $h_{21}$ has an equal opportunity to fetch assigned pieces.

Our solution to this issue is *prioritized assignment*. Specifically, when assigning pieces to be downloaded, a client starts with servers with higher ranks. Algorithm 4 gives more details on how pieces to be downloaded are assigned to servers.

**Algorithm 4**: updateAssignment()

```
// sort serverList by rank$_h$ (retrieved from manifest xml)
// and then $w_h$ if same rank;
for h ← serverList[0] ... serverList[serverList.size() - 1] do
    /* avail$_h$ is h's availability for assignment, see Algorithm 2 */
    if avail$_h$ then
        /* request from $h$ for up to $w_h$ pieces */
        reqPieceSet$_h$ ← $\emptyset$;
        /* playPoint: the "oldest" piece being consumed by player */
        i ← playPoint + 1;
        while i < reqLimit and reqPieceSet$_h$.size() < $w_h$ do
            if pieceStatusMap[i] = NOT_ASSIGNED then
                reqPieceSet$_h$ ← reqPieceSet$_h$ $\cup$ {piece i};
                pieceStatusMap[i] = ASSIGNED;
            i++;
        if reqPieceSet$_h$ != $\emptyset$ then
            avail$_h$ ← false;
            asyncHTTPRequest(h, reqPieceSet$_h$, timeout(T));
```

**Comparison with TCP**: Our window-based downloading adaptation algorithm is different from traditional TCP congestion control. (1) To maintain client QoE, it requires (at application-level, during $T$ seconds) that the total downloading rate across all servers to be at least video encoding rate. In particular, the sum of window sizes should satisfy $\sum_h w_h \geq \omega \cdot T$. The adaptation algorithm enforces this by setting initial window size for a primary server (*e.g.*, $w_{h11}$) to be $\omega \cdot T$, and for each backup server to be 1. (2) Different from TCP's per-segment window update, we apply AI on the window size after all requests of the entire window are completed. In steady state (client streaming smoothly), this allows the client to slowly probe higher-ranked servers. Upon primary server failure or congestion, the AI strategy increases the backup server's window size; due to self clocking and before reaching the streaming rate, the increase behavior is similar to TCP slow start, which is fast to allow request queue cleanup. (3) Although the algorithm maintains a window size for each server, it does not open a (TCP) connection to a lower-ranked server until necessary. Also, when the higher-ranked servers have sufficient capacity, the adaptation algorithm disconnects the lower-ranked servers.

**Active multi-rate streaming**: Just as CMO can be extended to handle multi-rate streaming, the preceding active clients can also be extended to take advantage of multi-rate. Specifically, consider a video encoded in a set of rates, say 1x, 2x, and 4x. As we derive Algorithm 3 and 4 from Figure 7, after a publisher defines its preference on video rates and the extended control diagram, we can similarly derive a control algorithm. As a simple example, a content publisher with a control goal of providing the highest video rate whenever possible will lead to a transition diagram that more servers are used until combined capacity is above 4x, if possible.

# 8. EVALUATIONS

In this section, we evaluate the cost and performance of our system design for content multihoming. In specific, we implement and test our optimization algorithm (CMO), the client adaptation algorithm, and the interactions between the two system components. We use real data to drive the run of our optimizer, and instrument clients on PlanetLab to conduct experiments.

## 8.1 Evaluation Methodology

**Content publishers**: We evaluate our algorithms using real traces of content requests to two production Video-on-Demand (VoD) publishers. We name the content publishers CP1 and CP2 respectively. We consider each video as a content object, and collect the following information about each video: its size, and the number of times that it is requested from each city per month for a 6-month duration. Table 4 shows the summary statistics of the content objects. Figure 8 plots detailed statistical distributions of object sizes ($s_i$), number of requests ($n_i$) to each object, and traffic volume ($s_i n_i$) of each object. These distributions are long-tail distributions.

We use the MaxMind GeoIP database to map a client IP address in the trace to a location area. Our evaluations define location areas as following: we start with each country as a location area; for a country with a large geographical span, we refine it to a next level; for example, we define each state in USA as a location area.

| | # Objects | Sum of Obj Size | Total Traffic | #Request |
|---|---|---|---|---|
| CP1 | 667,856 | 71 TB | 27,307 TB | 390,235,440 |
| CP2 | 529,411 | 40 TB | 12,114 TB | 153,129,348 |

**Table 4: Summary statistics of content objects.**

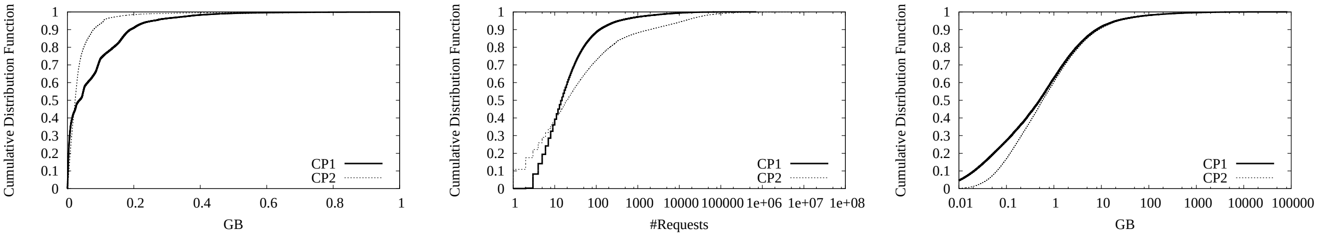
**Content distribution networks**: Our evaluation is based on three commercial CDNs: Amazon CloudFront, MaxCDN, and an anonymous private CDN which we refer to as CDN3. The geographic footprints of CloudFront and MaxCDN are shown in Figure 1, and the real charging structures and parameters of CloudFront and MaxCDN are shown in Figure 2. The server distribution and detailed price information for CDN3 are not shown due to privacy. Moreover, in our evaluations, we require that $p^a_{i,k} \geq 90\%$ for each CDN $k$ and each location object $i^a$.

To obtain how each CDN maps a location area to its charging region, we deploy a measurement client on each one of $536$ available PlanetLab machines to request objects from each CDN. We use `traceroute` to determine the locations of the CDN servers, as the GeoIP database can be inaccurate, *e.g.*, all CloudFront servers are always classified as in Seattle, WA, US. After computing the charging region intersections of the three CDNs, we pick the top 5 intersections that contain the most traffic. Table 5 shows the percentages of traffic to major geographical regions.

| | US | EU | SA | Asia & Pacific | Japan |
|---|---|---|---|---|---|
| CP1 | 19 % | 7 % | 1 % | 71 % | 2 % |
| CP2 | 77 % | 11 % | 6 % | 5 % | 1 % |

**Table 5: Traffic distribution across major geo regions.**

**Optimizer**: Our publishing optimizer is simple to implement (∼2000 lines of C++ code) and runs on a commodity PC with 2 quad-core Intel Xeon 2.33 GHz CPUs and 3 GB of memory. It takes about 12 minutes for CMO to compute the optimal assignments for each charging period (one month) for our data set.



(a) object size (b) number of requests (c) traffic volume

**Figure 8: Statistics of object size, number of requests to each object, and traffic volume for each object.**

**Active clients and controller**: We integrate our client adaptation algorithm into an Adobe Flash video player. Our player is deployed on CDN3 and can be accessed by any web browser on the Internet, including PlanetLab nodes. We leverage HTTP `range request` and Adobe Flash's `NetStream.appendBytes` (supported by version 10.1 and above) to integrate multiple CDN servers for one video streaming session. Our player periodically reports back through HTTP to a publisher controller. The reporting process allows the controller to obtain player IP, player local DNS server address, and the IP addresses of each CDN serving the player's content request. An implementation issue of the player is that the Flash browser sandbox does not allow UDP DNS queries. Hence, although a CDN typically resolves a DNS name to multiple server IPs, the player can have access to only one. We use two approaches to obtain backup server IPs for a CDN. The first is that the controller returns, in the manifiest file, additional server IPs reported recently by other closeby (same location area) players. The second is that the player makes a TCP connection to its LDNS or CDN DNS server (if allowed by Flash Socket Policy) to query DNS directly.

We install Mozilla Firefox with Adobe Flash Player (using Xvnc as XServer) on $412$ PlanetLab nodes and some personal laptops with ADSL and WiFi, and instrument these clients to conduct video streaming experiments. Note that although most of our clients are PlanetLab nodes which typically have good network connectivity rather than real clients, we control the settings to reflect two key factors that contribute to real client QoE degradation: server overload and network bottlenecks. Both factors are included in our stress tests to impose challenging, oscillating limits on bandwidth.

Specifically, based on our traces, we deploy active clients on PlanetLab according to their availability and geographical locations. We select random videos from CP1 and generate active client request load according to video request patterns of CP1.

**Content deployment**: We deploy video objects testing active clients on both CloudFront and CDN3. CloudFront has the best performance according to our PlanetLab measurement (see Table 2). We are able to run Adobe Flash Socket Policy service on CDN3, and hence active clients can use customized and optimized TCP sockets for requests to this private CDN. We also run multiple pre-tests (before the evaluation) to warm up both CDNs, *i.e.*, the edge servers prefetched the video content before the experiments start.

**CDN server capacity failure models**: We evaluate the effectiveness of our client adaptation algorithm under both controlled stress tests and real server congestions. In the real experiments using two CDNs and PlanetLab clients, we do *not* inject any failures, but the CDN servers can get temporarily congested due to the bursty nature of client request load.

**Performance metrics**: We evaluate both the *CDN cost* and *client QoE* of our content multihoming optimization. CDN cost is simply the total charge (in USD) by all CDNs given a CDN assignment, *i.e.*, the value of function $C(\{x^a_{i,k}\})$ defined in problem **Q** in Section 5.1. For client QoE, we use three performance metrics: (1) *freezes*, the frequency (number of times per view) a viewer encounters rebuffering during a video view, excluding cases due to initial start or user drag (seek). As observed in [10], freezes are a major factor reducing viewer's QoE. (2) *smoothness ratio*, the percentage of the clients that never encounter freeze. This is a statistical performance metric. (3) *buffering time*, viewer visible buffering time (in seconds) per video view, including startup delay and seek delay.

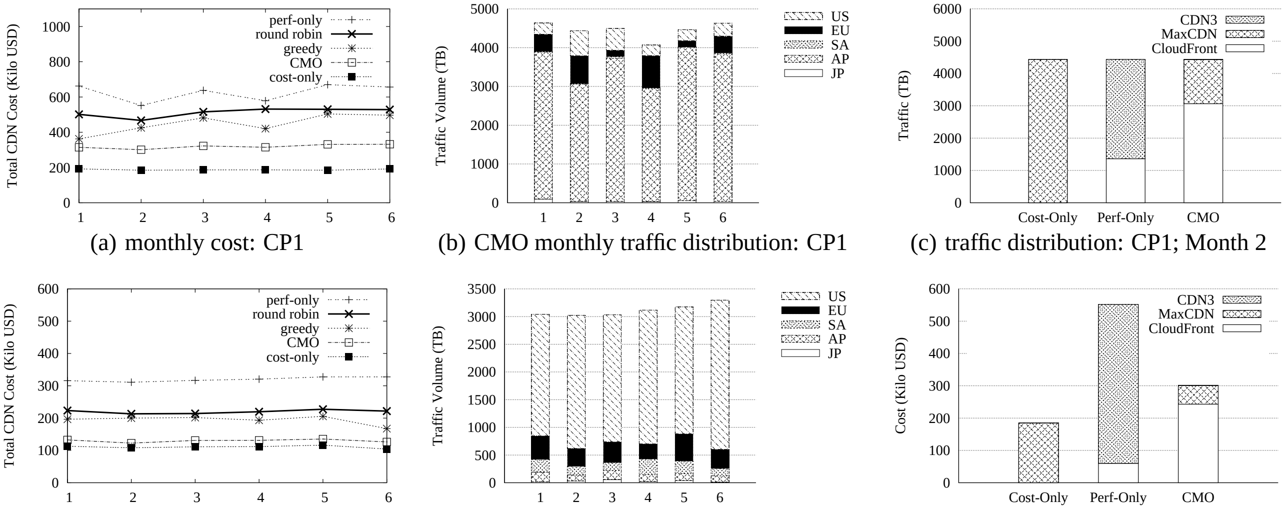
**CDN assignment algorithms**: We evaluate 5 CDN assignment algorithms: (1) *CMO*: This is the CMO algorithm defined in Section 6.2 considering both traffic and per-request cost defined in Section 6.3; (2) *greedy*: This algorithm assigns location objects sequentially in a uniformly random order. When assigning the next object, the algorithm computes the cost to be reached when the object is put in each feasible CDN. The object is assigned to the CDN resulting the lowest cost among the alternatives; (3) *round-robin*: This algorithm also assigns location objects sequentially in a uniformly random order. A CDN index is maintained. When assigning the next object, the algorithm uses round-robin, starting from the current CDN index, to assign the object to a feasible CDN; (4) *cost-only*: This algorithm minimizes cost without any performance considerations. (5) *perf-only*: This algorithm selects the best performing CDN(s) at each location area regardless the cost.

## 8.2 Publishing Cost Optimization

We start by evaluating the CDN cost savings of our CMO algorithm. At the beginning of a month, for each location area $a$, we use the content traffic to $a$ in last month as the traffic prediction in this month; we leverage the algorithms listed in Section 8.1 to decide how to redirect requests from various locations to the three CDNs; we then use the real traffic in the month to calculate the total monthly cost of CP1 and CP2.

We first compare the monthly costs to the two content publishers using CMO *vs* those using the other 4 algorithms. Figures 9(a) and 9(d) show the results. We observe that CMO saves around 30% $\sim$ 40% each month for both CP1 and CP2, compared with all algorithms that should satisfy real performance constraints (perf-only, round-robin, and greedy). For CP2, CMO's cost is close to cost-only because (1) the traffic of CP2 is mostly in US/EU, and (2) all 3 CDNs have good performance in US/EU and hence each can be used. Note that CMO achieves savings despite traffic fluctuations across billing periods. Figures 9(b) and 9(e) show the traffic demands from different regions in different months. From Figure 9(b), we see that the demand from US changes from 4000 TB to 3000 TB in the first 2 months. Despite such fluctuations and hence prediction variations, CMO achieves substantial cost savings.

Next we look at some details on how CMO assigns traffic among CDNs. Figure 9(c) shows the traffic assignments for CP1, during the 2nd month. As comparisons, we also show results for two simple extreme strategies: perf-only and cost-only. We observe that cost-only assigns all traffic to MaxCDN, but as we have seen in Table 2, this can lead to performance violations at locations such as China and Brazil. The perf-only strategy assigns a large fraction of traffic to CDN3, whose performance is the best in AP and JP. However, the cost of perf-only can be much higher. Specifically, we can see from Figure 9(f) that the cost of perf-only for CP1 during month 2 is $1.84$ times that of CMO. CMO assigns most traffic to



(a) monthly cost: CP1 (b) CMO monthly traffic distribution: CP1 (c) traffic distribution: CP1; Month 2

(d) monthly cost: CP2 (e) CMO monthly traffic distribution: CP2 (f) monthly cost: CP1; Month 2

**Figure 9: Cost and traffic distributions in the $6$ months with different CDN assignment algorithms.**

CloudFront and MaxCDN, as they satisfy performance constraints at lower costs.

## 8.3 Active Clients

Passive clients cannot handle the failures or congestions of the primary CDN, and hence may encounter QoE degradation at times. We demonstrate that active clients can protect per-view QoE despite CDN server failures/congestions.

**Stress tests**: We start with stress tests when delivering a 1080p HD video object encoded at $480$ KB/s. We run two sets of experiments: (1) only one CDN (primary), which has two servers named `primary1` and `primary2`; (2) two CDNs (one primary+one backup), each with one server, named `primary1` and `backup1` respectively. In each set, we vary the capacity of `primary1` in the following three cases: (1) step-down, in which the capacity of `primary1` is reduced down to only $10\%$ of video encoding rate and then recovers after 2 min; (2) ramp-down: in which the capacity of `primary1` is linearly decreased (to $10\%$ of the video encoding rate) in one minute and then linearly increased back; (3) oscillation, in which the capacity of `primary1` periodically falls down (to $10\%$ of the encoding rate) and then recovers after $20$ seconds. We plot detailed downloading rates to observe more behaviors. Figure 10 plots the results.

We make multiple observations on client behaviors. First, in all 6 cases, the client downloads at full speed at the beginning to build the required 4-second video buffer before playback can start. After playback starts, the active client continues to maintain a 16-second video buffer (total workload control).

Second, despite of `primary1`'s fluctuations, our active client achieves QoE protection by downloading from alternative servers. In (b) and (e), despite gradual `primary1` capacity changes, the aggregated downloading rate (labeled `total`) never falls below the streaming rate at any instance of time. In the other 4 cases, there are instantaneous dips when the total rate drops below the streaming rate. However, instantaneous dips may not lead to viewer visible freezes if the streaming buffer has enough data. In all 6 cases, the rebuffering wheel never appears during the entire video playback.

Third, the active client prefers `primary1` over `backup1`, following prioritized guidance. For example, in (d), `primary1` recovers at 250s and its utilization starts to increase, and at time 280s all requests have shifted from `backup1` back to `primary1`. One can also observe this "shifting-back" in (e) at time 160s-210s.

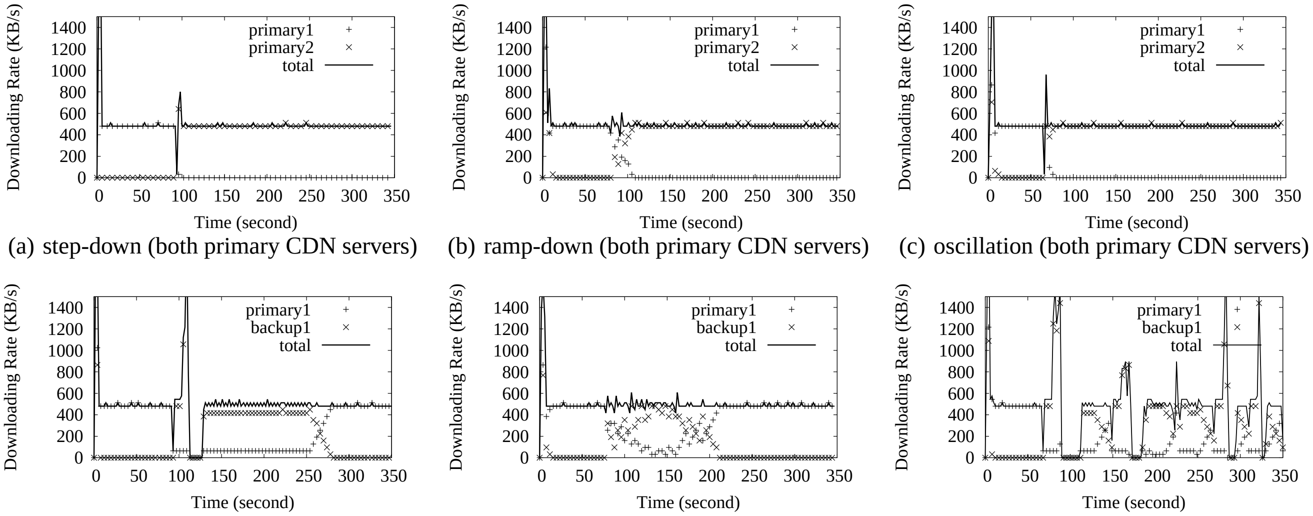
Fourth, the active client achieves "stickiness" for low session overhead. For example, comparing (a)(b) with (d)(e), we observe that in (a) and (b), there is no shifting-back to `primary1`, as `primary2` can handle the load alone and belongs to the same CDN as `primary1`.

Fifth, the active client performs relatively the worst in (f), when there is a *single* primary CDN server and the capacity of the server fluctuates widely. We observe multiple downloading spikes, as the client recovers from low rates resulted from HTTP request timeouts. The fluctuations of the client downloading rate reflects the primary server's capacity fluctuations. In practical deployment, it is recommended that a content publisher uses a primary CDN which offers *multiple* edge servers.

**PlanetLab experiments**: We next evaluate the statistical performance for both passive and active clients with real PlanetLab experiments. First, we measure smoothness. Figure 11(a) shows that with passive clients, $8\%$ clients experience video freeze. Thus, the smoothness ratio is $92\%$, which is considered as high performance in industry. Active clients improve viewer QoE and reduce the percentage of clients observing video freeze to only $3.8\%$. Thus, active clients reduce QoE degradation by $51\%$. We also calculate that active clients reduce the average number of video freezes from $4.78$ (for passive clients) to $2.19$. Second, Figure 11(b) shows the buffering time performance. Active clients reduce the average buffering time from $9.6$ seconds to less than half at $4.5$ seconds.

**Cost impact of active clients**: Active clients might increase publisher cost for two reasons. First, the optimization algorithm considers the impact of active clients by predicting the amount of traffic shifted to backup. Reality may be different from the prediction. Second, for simplicity, during evaluations, our optimization algorithm does not consider the additional number of requests due to backup protection.

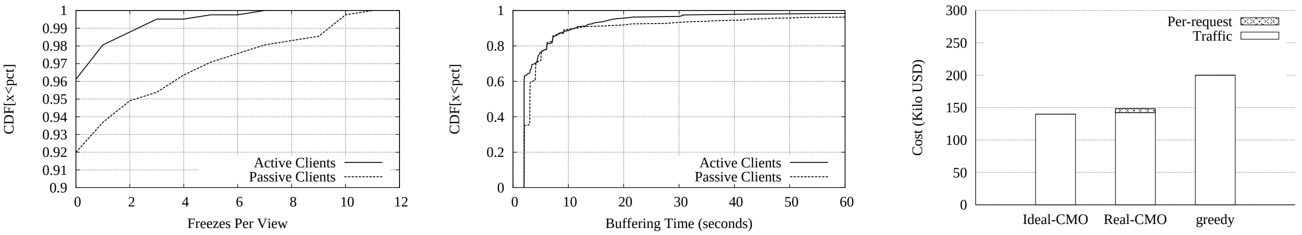
We evaluate the cost impact of client adaptation by comparing the computed optimal cost, the real cost during our PlanetLab experiments (after scaling up the traffic), and the cost of using round-robin CDN assignment. To better understand this impact, we further break down the cost into "traffic cost" and "per-request cost". Figure 11(c) shows the result. We observe that the cost impact in total is less than $5.6\%$ ($\sim 8,000$ USD added to $\sim 142,000$). Traffic deviation is less than $2.1\%$ from prediction and contributes to $1.4\%$ of the total $5.6\%$ difference. The additional number of requests accounts for $4.2\%$ of the total cost.



(a) step-down (both primary CDN servers) (b) ramp-down (both primary CDN servers) (c) oscillation (both primary CDN servers)

(d) step-down (primary and backup CDN) (e) ramp-down (primary and backup CDN) (f) oscillation (primary and backup CDN)

**Figure 10: Stress tests of client adaptation in CDN server failure cases.**



(a) freeze per view statistics (b) buffering time per view statistics (c) cost impact of client adaptation

**Figure 11: Per-view QoE in PlanetLab experiments.**

# 9. CONCLUSIONS

In this paper, we have conducted the first systematic study on content multihoming, by introducing the CMO algorithm and the client adaptation algorithm to optimize both the cost and the performance for content multihoming. Our realistic evaluations show that our content multihoming algorithms reduce publishing cost by up to $40\%$, and reduce viewer QoE degradation by $51\%$.

**Acknowledgments:** The research of Y. Richard Yang is supported in part by grants from NSF, and by a gift from Huawei. We are grateful to Aditya Akella (paper shepherd), Nicole Shibley, Peng Zhang, Haiwei Xue, Andreas Voellmy, Yin Zhang and SIGCOMM anonymous reviewers for valuable suggestions.

# 10. REFERENCES

[1] 01box. `http://cdn.01box.net`.

[2] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang. Unreeling netflix: Understanding and improving multi-CDN movie delivery. In *IEEE INFOCOM'12*.

[3] H. A. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. Van Der Merwe. A practical architecture for an anycast CDN. *ACM Trans. Web*, 5(4):17:1–17:29, Oct. 2011.

[4] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel. Finding a needle in haystack: Facebook's photo storage. In *USENIX OSDI'10*.

[5] G. Bertrand, E. Stephan, G. Watson, T. Burbridge, P. Eardley, and K. Ma. Use cases for CDNi. IETF Draft, Jan. 2012.

[6] D. Bertsekas. *Convex Analysis and Optimization*. 2003.

[7] CDN expert. `http://cdnexpertonline.com/node/45`.

[8] Cisco Systems. Cisco Visual Networking Index: Forecast and Methodology, 2011-2016.

[9] Conviva. `http://www.conviva.com`.

[10] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. In *ACM SIGCOMM'11*.

[11] Dyn CDN manager. `http://dyn.com/`.

[12] Geo best-of YouTube. `http://geobestofyoutube.gmapify.fr/`.

[13] D. Goldenberg, L. Qiu, H. Xie, Y. R. Yang, and Y. Zhang. Optimizing cost and performance for multihoming. In *ACM SIGCOMM'04*.

[14] A. Ioffe and V. Tihomirov. *Theory of Extremal Problems*. Elsevier Science Ltd, 1979.

[15] R. Krishnan, H. V. Madhyastha, and etc.. Moving beyond end-to-end path information to optimize CDN performance. In *ACM IMC'09*.

[16] Level 3 Intelligent Traffic Management. `http://www.level3.com/~/media/Assets/brochures/brochure_intelligent_traffic_management.pdf`.

[17] Limelight Traffic Load Balancer. `http://www.limelight.com/traffic-load-balancer/`.

[18] H. H. Liu, Y. Wang, Y. R. Yang, H. Wang, and C. Tian. Optimizing cost and performance for content multihoming. Technical Report YaleCS-TR1456, May 2012.

[19] MetaCDN. `http://www.metacdn.com/`.

[20] B. Niven-Jenkins, F. L. Faucheur, and N. Bitar. Content distribution network interconnection problem statement. IETF Draft, Jan. 2012.

[21] OnePica. `http://www.magentocommerce.com`.

[22] R. S. Peterson and E. G. Sirer. Antfarm: efficient content distribution with managed swarms. In *NSDI'09*.

[23] R. S. Peterson, B. Wong, and E. G. Sirer. A content propagation metric for efficient content distribution. In *ACM SIGCOMM'11*.

[24] I. Poese, B. Frank, B. Ager, G. Smaragdakis, and A. Feldmann. Improving content delivery using provider-aided distance information. In *IMC'10*.

[25] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *ACM CCS'09*.

[26] N. H. Sleumer. Output-sensitive cell enumeration in hyperplane arrangements. *Nordic J. of Computing*, 6:137–147, June 1999.

[27] XDN. `http://www.xdn.com`.