(Cache) Congestion Control

Kian Bahadori*
EPFL
Switzerland

Mahdi Hosseini*
EPFL
Switzerland

Elie Bruno
EPFL
Switzerland

Muhammad Abdullah
EPFL
Switzerland

Pavlos Nikolopoulos
EPFL
Switzerland

Katerina Argyraki
EPFL
Switzerland

Abstract

We hypothesize that the next Internet bottleneck will be shared cache space: when an end-user downloads content from the Internet, the dominant element that determines the speed and cost of the transfer will not be limited bandwidth (shared by multiple flows), but limited cache space (shared by multiple content providers). If this is true, then cache allocation should play a role akin to congestion control—jointly optimize for efficiency *and fairness*. We ask: Would fairness across content providers necessarily destroy cache efficiency? Are there realistic scenarios where fairness is efficiency-friendly?

CCS Concepts

• **Networks** → **Network resources allocation**;

Keywords

Edge caching, cache allocation, fairness, resource allocation

ACM Reference Format:
K. Bahadori, M. Hosseini, E. Bruno, M. Abdullah, P. Nikolopoulos, and K. Argyraki. 2026. (Cache) Congestion Control. In *The 25th ACM Workshop on Hot Topics in Networks (HotNets '26), November 16-17, 2026, Salt Lake City, UT, USA*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3847290.3859932

1 Introduction

We (networking people) care about congestion control, because it solves a quintessential networking problem: there is a shared network of finite bandwidth; there are its customers, the flows, which may demand more bandwidth than the network can supply; how to allocate the available bandwidth to the flows *efficiently* and *fairly*? Informally speaking, efficiency captures how well the network is used, while fairness captures how evenly the benefits of the network are distributed among the flows. In the late 90s, Kelly et al.[13] recast fairness as customer-utility optimization: they expressed the benefit gained by each customer (flow) as a utility function of the shared resource (bandwidth); and showed that, when the utility function is logarithmic, proportional fairness among customers emerges naturally from optimizing aggregate customer utility.

Imagine Internet congestion control that ignores customer utility and targets only efficiency. It does not make sense—there exist maximally efficient bandwidth allocations where some flows receive 0 bandwidth. We (as a community) have debated what the right fairness metrics are (utility functions) [13, 15, 20]; and whether congestion still occurs in the wild [27]. However, there is no question that when it does occur—when multiple network customers contend for bandwidth—*some* form of customer utility is an essential part of the solution [13, 15, 17, 18, 26].

In this paper, we hypothesize that the next Internet bottleneck is shared cache space: when an end-user downloads content from the Internet, the dominant element that determines the speed and cost of the transfer is not limited bandwidth (shared by multiple flows), but limited cache space (shared by multiple content providers).

This is not an unreasonable hypothesis: Today, when an end-user downloads content from the Internet, that typically comes through a cache located close to the end-user [23]. If the requested content is in the cache, it may reach the end-user within a few msec. If not, it needs to be fetched from an origin server that may be tens or even hundreds of msec away, potentially through expensive transit links [28]. There do exist big content providers that maintain their own caches or buy dedicated cache space from cache operators, however, we expect that the vast majority of content providers buy shared caching services (§2). So, cache contention can significantly affect content providers: the cost per transferred content byte, as well as the quality-of-experience (QoE) with which the end-user views the content [3].

If our hypothesis is true, then cache allocation should be understood as joint optimization of efficiency and customer

*These authors contributed equally to this work.



This work is licensed under a Creative Commons Attribution 4.0 International License.
HotNets '26, Salt Lake City, UT, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-3029-0/2026/11
https://doi.org/10.1145/3847290.3859932

utility—much like congestion control [13, 15, 17, 18, 26]: there is a shared cache of finite space; there are its customers, the content providers, which may demand more cache space than the cache can supply; how to allocate the available cache space to the content providers *efficiently* and *fairly*?

The reader may question that fairness is relevant to modern caches, thinking that each content provider gets the cache space that it pays for. However, we expect that most content providers today buy a shared-caching plan that does *not* guarantee any amount of cache space (§2). Recent work [3] summarized the situation as follows: Consider two content providers that supply similar content to audiences in the same geographical areas; suppose they buy the same shared-caching plan from the same cache operator; in principle, the content provider with more popular objects will enjoy a higher hit rate—and bigger cache footprint—at the same or even lower cost per transferred byte.

The reader may still balk at the idea: caches by design prioritize popular content; how could a cache allocation be "fair"? Dehghan et al. [8] argued that a given increase in hit rate may bring different value to different content; and they elegantly recast fairness across different content as utility optimization. Since content providers are the natural fairness actors in a shared cache, we can adjust their logic and cast fairness across content *providers* as utility optimization. Thus, a "fair cache allocation" would assign hit rates to objects so as to maximize the aggregate utility of content providers.

Today, cache operators ignore content-provider utility. Their cache allocation typically relies on Least Recently Used (LRU) [6, 10] or other heuristics that try to increase the aggregate hit rate of the cache [5, 7, 19, 24, 25]. One can view this as increasing cache efficiency, or increasing the utility of the cache operator—either way, it ignores the utility of the content providers.

We ask: if shared caches are the next Internet bottleneck; and we view cache allocation as joint optimization of cache efficiency and content-provider utility; does one objective always destroy the other? Are there realistic scenarios where fairness is efficiency-friendly?

We make three points: LRU and aggregate hit-rate optimization are only point solutions in a much richer design space; there exist cache allocations that are nearly as efficient yet substantially fairer than those achieved with conventional popularity-based algorithms; these efficient-yet-fairer allocations can be realized by simple online controllers. Bottom line: we argue that modern caches can be fairer to their customers without sacrificing their efficiency.

2 Context

Consider a content provider that benefits from caching (e.g., a streaming service or a social network). When an end-user requests a piece of content from this content provider, the request is directed to a user-facing cache, typically located within or close to the end-user's ISP [22, 23]. If a request misses in the user-facing cache, it is typically forwarded through one or more upper-tier caches before reaching the origin [1, 2]. The last upper-tier cache before the origin typically plays the role of "origin shield," i.e., it protects the origin from traffic spikes by aggregating lower-tier requests [4, 11].

There exist at least three common scenarios: (a) The content provider relies on caching and storage infrastructure that belongs to itself or a parent company (e.g., Meta, YouTube, Prime Video). (b) The content provider buys caching and storage services from different companies. In this scenario, the network path between last-tier cache/origin shield and origin may traverse expensive transit links. (c) The content provider buys caching and storage services from the same company (e.g., CloudFront caching and S3 storage from Amazon) [4].

Charging models—for scenarios (b) and (c)—differ but they have a common base: the content provider pays a fixed fee per byte requested from the user-facing cache, whether it is a hit or a miss. In scenario (b), in case of a miss, the cost of the content bytes transiting from origin to upper-tier cache ("egress tax") is typically absorbed by the content provider. A content provider may also pay for differentiated caching services, e.g., choosing the location of their origin shield (can reduce egress tax by improving the hit rate of the origin shield); and—in the case of one cache operator [2]—dedicated upper-tier cache space.

In this landscape, we expect that the vast majority of content providers share common caches and will continue to do so in the foreseeable future. Dedicated cache deployments are unlikely to be economically viable for most providers: demand is fragmented across regions, popularity shifts over time, and operators multiplex storage across many tenants.

In the rest of the paper, we take it as a given that shared caches will be the "new Internet bottleneck" and look at cache allocation through this lens. We consider a single-tier cache shared by a set of content providers. This ignores the potentially significant interplay between lower- and upper-tier cache behavior. However, it is the right setting for making the fundamental observations we want to discuss.

3 Efficiency and Fairness

Consider a cache of capacity C, shared by a set $\mathcal{I}$ of content providers. Content provider i offers a set of objects O_i. Object $o \in O_i$ has size s_o and long-run average request rate λ_o at this cache (i.e., λ_o is the number of requests for o divided by the observation time in the long run). Let content provider i's aggregate long-run request volume be $\Lambda_i \triangleq \sum_{o \in O_i} \lambda_o$.

Now consider a cache allocation where object o attains a long-run hit rate $h_o \in [0, 1]$ (i.e., h_o is the fraction of o's

requests that hit). The hit rate of content provider i is the request-weighted average of the hit rates of its objects:

$$h_i = \sum_{o \in O_i} \frac{\lambda_o}{\Lambda_i} h_o. \tag{1}$$

Note the impact of content popularity: improving the hit rate of a heavily requested object contributes more to the provider-level hit rate than improving the hit rate of a rarely requested object.

Model. We assume that an object's long-run hit rate h_o equals the fraction of time it occupies the cache. This assumption links hit rate to cache residency and enables us to formulate a feasibility constraint in terms of hit rates: we consider a cache allocation feasible when $\sum_{i \in \mathcal{I}} \sum_{o \in O_i} s_o h_o \le C$, i.e., the *average* cache footprint fits in the cache. By reasoning about average—as opposed to instantaneous—footprint, we abstract away which objects are in the cache at each instant. Hence, we can perform our analysis without modeling temporal request patterns or committing to any specific cache replacement policy.

This model can be too strict or too permissive. For instance, consider two objects, each of size C, and an allocation giving them hit rates 70% and 50%. The model rejects this (because the average footprint exceeds C), yet if the two objects are popular during disjoint time intervals, both may attain these hit rates. Conversely, an allocation of 60% and 40% is feasible under the model, yet whether the two objects attain these hit rates depends on how request arrivals interact with the cache replacement policy.

Nevertheless, this model is useful: It isolates two fundamental aspects of popularity—provider request volumes and within-provider popularity skew—that shape the efficiency/fairness trade-off even in the presence of temporal request patterns and under diverse cache replacement policies. We return to the model's limitations in §4. We introduce a practical cache replacement policy (that depends on the model only weakly) in §§5 and 6.

We now consider the following problem: The cache operator's obvious objective is efficiency—serve as much traffic as possible from the cache. What if the cache operator also wants some notion of fairness across content providers? Would such an objective always destroy efficiency?

Cache efficiency. Each byte should prevent as many (expensive) misses as possible. The most general objective would weigh hits by the cost they avoid. If serving object o from the cache instead of the miss path saves κ_o, then the cache operator wants to maximize $\sum_{i \in \mathcal{I}} \sum_{o \in O_i} \lambda_o \kappa_o h_o$. To simplify things, suppose that all avoided misses have the same value for the cache operator. Then cache efficiency is simply aggregate hit volume:

$$H_{\text{tot}}(\mathbf{h}) \triangleq \sum_{i \in \mathcal{I}} \sum_{o \in O_i} \lambda_o h_o. \tag{2}$$

This gives a clean efficiency benchmark: Maximize aggregate hit volume

$$H^{\star}_{\text{eff}}(C) = \max_{\{h_o\}} \sum_{i \in \mathcal{I}} \sum_{o \in O_i} \lambda_o h_o \tag{3}$$

subject to the cache capacity constraint

$$\sum_{i \in \mathcal{I}} \sum_{o \in O_i} s_o h_o \le C, \text{ and } 0 \le h_o \le 1.$$

This is not a claim about what LRU, FIFO, or any particular replacement policy achieves. It is only the *ideal* cache allocation that maximizes aggregate hit volume under the cache capacity constraint. With equal-size objects, it caches the most requested objects first.

Content-provider fairness. A cache hit may matter to a content provider for different reasons. It may improve QoE: lead to lower startup delay, fewer rebuffering events, fewer video-bitrate switches, lower page-load time [21]. It may reduce miss-related cost: lead to fewer origin fetches, less paid egress, fewer backhaul resources exercised. It may also improve robustness during overload or flash crowds. Eventually, all of this may translate into business value: watch time, churn, ads, conversions, etc. [9, 14].

We do not model these effects separately. Instead, we subsume all of the provider's cache-induced benefits into a net-value function that depends only on the provider's hit rate:

$$V_i(h_i),$$

and assume that it is non-decreasing in h_i. The functions V_i may differ across providers: two providers may receive the same hit rate and value it differently; or they may receive different hit rates and derive the same value. For example, a provider whose origin is far away, expensive, or fragile may value an extra cache hit more than a provider whose miss path is already cheap and fast.

A fairness rule can now be viewed as specifying how provider values should be compared or constrained [15, 20]. For example, it may require equal values, bounded value gaps, minimum values, proportional fairness, max-min fairness, or something else. Once such a rule selects a particular *target* vector, $[V_i^{\star}]_{i \in \mathcal{I}}$, and the functions V_i are invertible, the target values imply provider hit-rate targets

$$h_i^{\star} = V_i^{-1}(V_i^{\star}), \qquad i \in \mathcal{I}. \tag{4}$$

Efficiency *and* fairness. Now let $\mathcal{F}$ be the set of all capacity-feasible allocations, and let $\mathcal{F}_{\text{fair}} \subseteq \mathcal{F}$ be the subset that also satisfies a chosen fairness rule. Because of the feasible-set inclusion principle, we have:

$$\max_{h \in \mathcal{F}_{\text{fair}}} H_{\text{tot}}(h) \le \max_{h \in \mathcal{F}} H_{\text{tot}}(h) = H^{\star}_{\text{eff}}(C).$$

Informally, adding a fairness constraint cannot improve efficiency. However, the inequality need not be strict. If an efficiency-optimal allocation already satisfies the fairness

rule, then fairness comes for free. And even when fairness loses against $H^{\star}_{\text{eff}}(C)$, it may still beat a practical cache replacement policy (like LRU) that happens to operate far from the efficiency-optimal benchmark. So: the relevant question is not whether fairness *may* reduce ideal efficiency—it may—but *when* the reduction is significant and when it is negligible.

4 When is fairness efficiency-friendly?

A preview of the answer is: whenever the content-provider target values required by fairness are "aligned" with how popularity is spread across and within content providers. Fairness asks for some set of values $V_i^{\star}$; whether the corresponding hit rates $h_i^{\star}$ are good or bad for cache efficiency depends on which content provider receives each value.

This "alignment" is determined in two steps:

First, target hit rate $h_i^{\star}$ may be on par with content provider i's overall popularity Λ_i or not. The aggregate hit volume attained by a realized fair target is given from Eqs. (1) and (2):

$$H_{\text{fair}} = \sum_{i \in \mathcal{I}} \Lambda_i h_i^{\star}, \tag{5}$$

which means that if provider i receives target $h_i^{\star}$, its realized contribution to aggregate hit volume is $\Lambda_i h_i^{\star}$. So, all else being equal, allocating a high hit-rate target to a large-volume content provider is better for aggregate hit volume than giving it to a small-volume provider. Whether this is achieved or not depends not only on the fairness rule, but also the shape and heterogeneity of the functions V_i: the fairness rule chooses values $V_i^{\star}$, while V_i^{-1} maps those values to hit rates.

Second, the cache must realize provider-level hit-rate targets through object-level allocations; even the same target hit rate $h_i^{\star}$ may be cheap or expensive to realize, depending on how requests are distributed across its objects (i.e. the popularity skew across its catalogue). For example, two providers may have the same Λ_i and the same target $h_i^{\star}$, and therefore contribute the same aggregate hit volume once their targets are realized. Yet, one may require much less cache than the other: if its catalog is concentrated, a few cached objects can deliver most of its hit rate; if its catalog is flat, many objects are needed.

To capture the latter step precisely, we define $B_i(x)$ as the minimum cache footprint needed to give content provider i hit rate at least x:

$$B_i(x) \triangleq \min_{\{h_o\}_{o \in O_i}} \sum_{o \in O_i} s_o h_o$$

subject to $\sum_{o \in O_i} \frac{\lambda_o}{\Lambda_i} h_o \geq x$, and $0 \leq h_o \leq 1$.

The structure of B_i is simple: To minimize footprint while targeting a given hit rate x, the cache should spend bytes on the objects that increase the provider hit rate the most per byte. Object o contributes λ_o/Λ_i to h_i per unit increase in h_o, and costs s_o bytes. Hence, objects should be ordered by $\frac{\lambda_o}{s_o \Lambda_i}$, or, equivalently, within a provider, by λ_o/s_o. With equal-size objects, this simply means caching the provider's objects in decreasing popularity order.

Eq. (5) may suggest that only the aggregate provider volumes Λ_i matter, but this is true only after the fair target has been realized. It says nothing about how much cache was needed to realize that target. That cost appears in the feasibility condition through $B_i(h_i^{\star})$: *A target vector $[h_i^{\star}]_{i \in \mathcal{I}}$ can be realized within the cache only when the minimum footprints fit: $\sum_{i \in \mathcal{I}} B_i(h_i^{\star}) \leq C$.* Among allocations that realize the same targets, using the cache-minimal footprints is the best case for cache efficiency; any additional footprint spent to obtain the same $h_i^{\star}$'s only consumes capacity that could have served other traffic. Thus, $\Lambda_i h_i^{\star}$ captures the aggregate-hit contribution of a realized target, while $B_i(h_i^{\star})$ captures the cache footprint required to obtain that contribution.

Conclusion. Two effects jointly determine whether fairness is efficiency-friendly: fairness is more efficiency-friendly when the induced hit-rate targets are larger for larger-volume providers; and when within-provider popularity makes those targets cheap to realize. These effects can compensate for one another rather than both being necessary: Sufficiently concentrated within-provider popularity can offset poor provider-level alignment. Conversely, even when catalogs are relatively flat and targets are expensive to realize, fairness can remain efficiency-friendly if larger-volume providers receive higher hit-rate targets, so that the cache space spent on them prevents more misses. Thus, the cost of fairness is not determined by the fairness rule alone; it depends on the full chain:

$$\text{fairness rule} \Rightarrow [V_i^{\star}] \Rightarrow [h_i^{\star}] \Rightarrow [B_i(h_i^{\star})] \Rightarrow H_{\text{fair}}.$$

Selecting an operating point. We want to illustrate our conclusion quantitatively—show that, under realistic assumptions, there exist feasible cache allocations that are both fair and efficient—almost as efficient as the allocations achieved by LRU. To do that, we need a mechanism for finding such cache allocations. We pick standard α-fair utility optimization [8, 13, 20] subject to a feasibility constraint, which we augment with an efficiency term. The exact formulation may not matter for this submission, but we still state it and define its parameters, so that the reader knows where the numbers that we show later come from.

We look for a cache allocation where object o has long-run hit rate h_o, such that:

$$\max_{\{h_o \in [0,1]\}} \quad \sum_{i \in \mathcal{I}} w_i u_\alpha\big(V_i(h_i)\big) + \beta \frac{\sum_{i \in \mathcal{I}} \sum_{o \in O_i} \lambda_o h_o}{\sum_{i \in \mathcal{I}} \Lambda_i} \tag{6}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{I}} \sum_{o \in O_i} s_o h_o \leq C.$$

The first term (before the +) captures fairness. Its parameters are straightforward: $w_i > 0$ is a content-provider weight, to encode contractual priorities or policy choices of the cache operator (e.g., content providers paying for "premium service" would receive a higher weight). $u_\alpha(x)$ is the α-fair utility function[1], where α controls how aggressively to "protect" content-providers with small net value: $\alpha = 0$ just maximizes the aggregate net value, $\alpha = 1$ enforces weighted proportional fairness, and larger α gives increasingly max-min-like behavior [8, 13, 15, 20, 26]. The second term captures efficiency[2], and β controls how much efficiency matters relative to fairness. Finally, the constraint approximates feasible allocations (the required cache footprints fit in the cache).

There may be other ways to find efficient-yet-fair cache allocations; we started from this one because it provides a generalized, tunable fairness objective (via α), while allowing efficiency to be incorporated into the same objective function. In addition, the overall formulation exposes marginal utility scores that make it suitable for practical caching algorithms, such as the one we implement in the following sections.

Beyond our steady-state analysis. A more explicit model would relate object hit rates to cache occupancy through the request process and the cache replacement policy. Consequently, the footprint function $B_i(x)$ would also depend on request timing, and neither the simple λ_o/s_o ordering nor the average-footprint-based feasibility constraint would generally suffice to characterize feasible allocations.

These complications do not remove the two effects identified above. Eqs. (1) and (5) still hold for long-run request rates and hit rates, regardless of request timing: increasing a large-volume provider's hit rate yields more aggregate hits than the same increase for a small-volume provider. Similarly, improving the hit rate of a popular object contributes more to its provider's hit rate than improving that of a less popular object. Thus, within-provider popularity still affects how cheaply the targets can be realized, although the required footprints also depend on request timing and whether providers need cache space at the same time.

All that said, our practical online controller (§5) reacts to actual requests and enforces physical cache capacity through insertions and evictions—not the average-footprint constraint; it uses the latter only when computing its marginal-utility score, which serves as a heuristic signal for reacting to each request. A more explicit model could provide a more informative score, but exploiting it online would require estimating temporal request patterns, rather than just average request rates.

[1] $u_\alpha(x) = \frac{x^{1-\alpha}}{1-\alpha}$ if $\alpha \neq 1$, and $u_1(x) = \log x$.

[2] It is the normalized aggregate hit rate $H_{\text{tot}}/\sum_i \Lambda_i$. For a fixed request workload $\sum_i \Lambda_i$, maximizing this is the same as maximizing H_{tot}. We use the normalized term, so that β controls a trade-off against a quantity whose scale does not change with workload size.

Table 1: Summary stats for both cases.

#Prov.	#Obj./Prov.	Baseline	Aggr. Hit Rate	Net-value CoV
1000	1	Efficiency Opt.	0.785	2
		Efficiency LRU	0.692	0.993
		Fairness Opt.	0.200	0
		Fairness TTL	0.316	0.118
20	50	Efficiency Opt.	0.754	0.296
		Efficiency LRU	0.624	0.344
		Fairness Opt.	0.651	0
		Fairness TTL	0.642	0.010

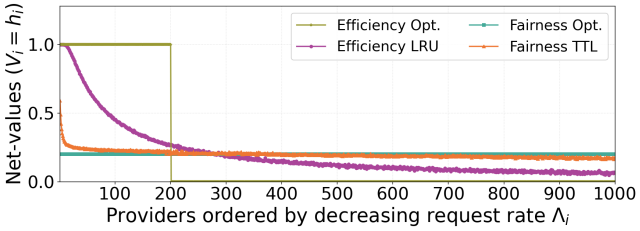


Figure 1: Simplest case: Provider net values.

5 A Simple (Bad) Case

Suppose that each provider has only one cacheable object; all objects have the same size s; all content providers share the same increasing and invertible net-value function.

We discuss this case because it is conceptually straightforward, and it clearly illustrates the challenge—how fairness can destroy efficiency: Because all content providers share the same invertible V, equal-value fairness implies equal hit-rate targets. When the distribution of request rates across objects is skewed (the common case in practice), equal hit rates unavoidably reduce cache efficiency significantly.

We implement this case in NS3, and compare the efficiency and fairness of 4 baselines:

(a) *Efficiency–optimal*: Ignoring fairness, it chooses the maximally efficient cache allocation. We find it by solving Eq. (3) using a standard convex solver Clarabel [12].

(b) *Fairness–optimal*: Chooses the most efficient cache allocation that also obeys the equal-value fairness rule (where all content providers have a common long-run hit rate). We directly compute it from the cache capacity and the number of providers $|\mathcal{I}|$ as follows:

$$h_i = h^\star = \min\left\{1, \frac{C}{(|\mathcal{I}|s)}\right\}, \qquad \forall i \in \mathcal{I}. \tag{7}$$

We expect this to yield poor efficiency: In the footprint notation of §4, since the object- and provider-hit rates coincide, we get $B_i(h^\star) = sh^\star,\ \forall i \in \mathcal{I}$. I.e., the fair allocation spends the same amount of cache on providers whose contributions to aggregate hit volume differ in proportion to Λ_i. To be more efficient, it should have done the opposite: cache first the most popular providers—the ones with large Λ_i—due

to Eq. (5). Hence, when capacity is scarce and the Λ_i's are heterogeneous, equal-value fairness inevitably loses hits.[3]

(c) *Efficiency–LRU* is standard LRU. Unlike the first two baselines (which compute offline a cache allocation by solving an equation/optimization problem), this is an implementable online algorithm: it updates its state (which objects are in the cache) upon each request arrival. We view it as a practical approximation of Efficiency–optimal.

(d) *Fairness–TTL* is an implementable online algorithm that we designed to approximate Fairness-optimal. It tries to solve Eq. (6) for $\beta = 0,\ \alpha = 1,$ and $w_i = 1$ through TTL updates[4]. We adapt the standard Lagrangian/primal-dual method of utility optimization, used in congestion control [13] and TTL caching [8], to our formulation (Eq. (6)): Upon the arrival of a request for object $o \in O_i$, we compute the marginal utility score[5]: $\text{Score}(o) = w_i u'_\alpha(V_i(h_i))V'_i(h_i)\frac{\lambda_o}{s_o\Lambda_i} + \beta\frac{\lambda_o}{s_o\sum_j\Lambda_j} - \mu$, where μ is the shadow price of the Lagrangian method (a quantity capturing the capacity constraint: it increases when the occupancy of unexpired objects approaches capacity and decreases otherwise). Positive score means that increasing object o's cache residency improves the objective more than it costs in cache space—hence o's TTL should increase; negative score means the opposite. For adjusting the TTL value, we use a lightweight Additive-Increase-Multiplicative-Decrease (AIMD) rule: if the score is positive we increase the object's TTL by 1sec; otherwise we decrease it by 50%. In our experiments, changing these AIMD constants did not change the qualitative long-run conclusions; it mainly affected convergence speed and TTL oscillations.

Experiment. 1000 content providers share the same cache, which can hold only 20% of the overall content; they also share the simplest net-value function $V(h) = h$ (i.e. provider net values are equal to their hit rates). Each provider has a single cacheable object; all objects have the same size; Λ_i's follow a Zipf popularity distribution with exponent $\zeta = 1$ and are normalized so that $\sum_i \Lambda_i = 100$ reqs/sec. Objects are requested by clients using UDP; the requests for each object i arrive at the cache as a Poisson process with mean rate Λ_i (per-object and per-provider request rates coincide here). Upon each request, the cache serves the object locally on a hit; on a miss, it fetches it from the origin and then applies the active insertion/TTL policy. This simplified setup abstracts away routing and congestion effects, so that the experiment isolates the behavior of the cache-allocation policies.

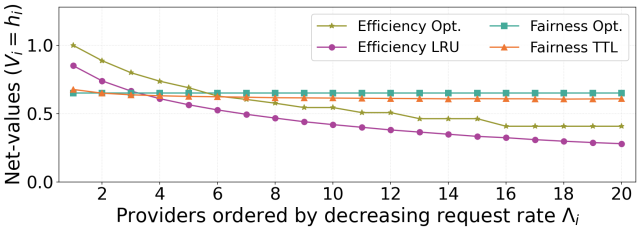


Figure 2: Interesting case: Provider net values.

Results. Figure 1 shows the long-run net value (equivalently hit rate) experienced by each content provider; each curve/color corresponds to a different baseline. For the two online baselines (Efficiency-LRU and Fairness-TTL) we report steady-state values measured after a warm-up period in which every object received at least 50 requests. Table 1 summarizes the same run using one efficiency metric, aggregate hit rate, and one fairness metric, the coefficient of variation (CoV) of provider net values (lower means fairer).

Efficiency–optimal spends the cache on the most popular content providers: Because the cache can hold only 20% of the objects, the benchmark gives hit rate 1 to the 200 highest-request-rate providers and hit rate 0 to the rest, modulo some rounding/fractional allocation at the boundary. This yields the maximum aggregate hit rate $H^\star_{\text{eff}}/\sum_i \Lambda_i = 0.785$.

Fairness–optimal gives all content providers the same hit-rate target, $h^\star = 0.2$. By Eq. (5), its normalized aggregate hit rate is therefore $\frac{H_{\text{fair}}}{\sum_i \Lambda_i} = h^\star = 0.2$, which is 74.5% below the efficiency benchmark. This loss is not due to the shape of V: if the same invertible V applies to all content providers, equal values imply equal hit rates whether V is linear, concave, or convex. It is a direct consequence of the first misalignment: cache is scarce, yet fairness assigns the same cache footprint to providers whose request rates Λ_i differ substantially.

Fairness–TTL lands at an intermediate operating point: it achieves higher aggregate hit rate than the Fairness—optimal, at the cost of nonzero provider inequality (net value CoV is 0.118 instead of 0), while still being substantially fairer than LRU. So, AIMD behaves more like a lightweight online approximation rather than an exact optimizer.

6 An Interesting (Good) Case

We now consider the same setting as in §5 but this time assume that each provider has a catalog of multiple objects. This models a more realistic caching paradigm: Video streaming providers may use the same shared edge cache to serve the same type of content, and each of them has a large catalog of videos. All such providers may share the same invertible value function $V(h)$ that maps hit rates to QoE scores (e.g.

[3]There exist trivial cases where fairness is free—all objects have the same popularity, or the cache can hold all objects—but we do not discuss those.
[4]TTL updates indirectly control the cache allocation without requiring explicit cache partitioning or centralized object placement. We use soft TTLs: expiry does not immediately evict an object; it may continue to serve hits until its slot is needed for another insertion. Whenever an insertion needs space, we evict the earliest-expired resident object, or the least recently used one if none has expired.
[5]To compute the score of each object o, the controller maintains EWMA-estimates of λ_o's and h_i's; we update these estimates upon each request.

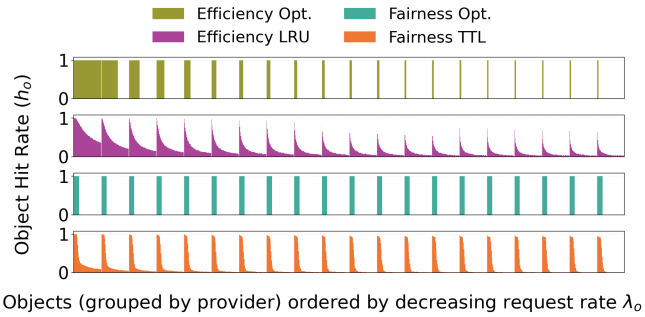


Figure 3: Interesting case: Object hit-rate allocations.

VMAF [16], MOS-style scores, or standardized models such as ITU-T P.1203/P.1204), and each provider has many objects.

Once providers have multiple objects, the cost of fairness depends not only on provider-level request volumes Λ_i, but also on how these volumes are distributed inside each provider's catalog. Equal-value fairness again requires equal provider hit-rate targets. However, unlike the one-object case, a provider can reach a high hit rate by caching only the popular prefix of its catalog. This creates an opportunity when popularity within each catalog is skewed: a small popular prefix can deliver a large fraction of each provider's hit rate. Thus the common fair target hit rate can be much higher than in the one-object case, even under the same total cache capacity.

Experiment. Consider again the same shared cache, same net-value function and the same overall number of cacheable objects as in §5, but the objects belong to 20 providers (50 objects each). The overall request volume is again $\sum_i \Lambda_i = 100$ reqs/sec, distributed as follows: provider-level request rates Λ_i follow a Zipf popularity distribution with exponent $\zeta = 1$, while object-level request rates within each provider i's catalog follow the same Zipf distribution, but normalized so that $\sum_{o \in O_i} \lambda_o = \Lambda_i$.

We compute the same baselines as in §5, with one difference: For Fairness–optimal Eq. (7) is not relevant. Instead, we use the proportionally-fair allocation found by Clarabel when solving Eq. (6) for $\beta = 0$, $\alpha = 1$, and $w_i = 1$ as an offline oracle. Under this setup, the proportional-fair solution coincides with the most efficient equal-value fair allocation (Fairness–optimal). This change should not be read as the source of the phenomena we observe. The behavior of Fairness–optimal still follows from the alignments discussed in §4; the optimization problem is simply the operational way to compute the most efficient equal-value allocation.

Results. Figure 2 illustrates how popularity skewness within catalogs makes fairness efficiency-friendly. Fairness–optimal now equalizes content providers at net-value/hit rate $h^\star = 0.651$—much higher than in the simple case. This, in turn, yields a much higher normalized aggregate hit rate $H_{\text{fair}}/\sum_i \Lambda_i = h^\star = 0.651$ (Eq. (5)), hence a much smaller efficiency loss 13.7%. Here, the two effects compensate (§4): equal hit-rate targets are poorly aligned with the unequal provider volumes, but concentrated within-provider popularity makes the common target cheap enough that Fairness–optimal remains competitive with LRU (0.651 vs. 0.624).

This finding is not incidental and can be understood analytically. Each provider has 50 equally-sized objects with the same normalized Zipf(1) popularity, and the cache holds 20% of all objects. Fairness–optimal therefore stores in the cache only the 10 (i.e. $50 \cdot 20\%$) most popular objects per provider, which yields: $h^\star = (\sum_{r=1}^{10} 1/r)/(\sum_{r=1}^{50} 1/r) = 0.651$. For LRU, the standard characteristic-time approximation [6, 10] estimates object o's hit probability as $1 - e^{-\lambda_o T_C}$, where T_C is chosen so that $\sum_o \left(1 - e^{-\lambda_o T_C}\right) = 200$, the cache capacity in objects. This predicts an aggregate hit rate of 0.633, close to the simulated value 0.624. More generally, in this symmetric-catalog setting, the fair hit rate depends on within-provider skew but not on provider-level skew, unlike LRU, whose hit rate also depends on the latter (via λ_o). This difference can allow Fairness-optimal to outperform LRU.

Figure 3 visualizes the same mechanism at the object level: Fairness–optimal concentrates cache residency on the popular objects within each provider and produces an allocation close to Efficiency–optimal, and much closer than Efficiency–LRU does. Even when all providers have the same normalized catalog shape (Zipf(1) here), within each provider, caching the most popular objects already captures a large fraction of requests; this also explains why the common hit-rate target was much higher than in the simplest case for the same cache capacity. More interestingly, Fairness–TTL not only achieves a better aggregate hit rate (0.642) than Efficiency–LRU (0.624), but does so with an order of magnitude lower provider inequality: CoV 0.010 instead of 0.344. This is not unexpected: with Fairness–TTL, popular providers receive larger shares at first, but as their marginal gain decreases, capacity shifts toward smaller providers, ensuring balanced provider net values.

We explored additional cases—object/provider combinations, non-Poisson/bursty requests, concave and/or asymmetric net-value functions, and TTL-update rules. While Fairness–TTL's performance varied with request patterns, our experiments broadly supported the insight from §4: favorable popularity alignment can make fairness efficiency-friendly, allowing it to match or even outperform LRU.

Bottom line: if shared cache space becomes—already is?—a dominant bottleneck in Internet communication, it makes sense for caching algorithms to incorporate notions of fairness; cache operators could do this without necessarily compromising their efficiency.

References

[1] 2024. Akamai pricing. https://techdocs.akamai.com/property-mgr/docs/tiered-dist#tiered-distribution-with-standard-tls-certificate

[2] 2026. Tiered Cache. https://developers.cloudflare.com/cache/how-to/tiered-cache/

[3] Muhammad Abdullah, Mughees Ur Rehman, Pavlos Nikolopoulos, and Katerina Argyraki. 2025. Edge Caching as Differentiation. In *Proceedings of the ACM SIGCOMM 2025 Conference*. 164–183.

[4] Amazon Web Services. 2026. Use Amazon CloudFront Origin Shield. https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/origin-shield.html Amazon CloudFront Developer Guide. Accessed: July 2026.

[5] Daniel S Berger, Ramesh K Sitaraman, and Mor Harchol-Balter. 2017. {AdaptSize}: Orchestrating the hot object memory cache in a content delivery network. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 483–498.

[6] Hao Che, Ye Tung, and Zhijun Wang. 2002. Hierarchical web caching systems: Modeling, design and experimental results. *IEEE journal on Selected Areas in Communications* 20, 7 (2002), 1305–1314.

[7] Asaf Cidon, Assaf Eisenman, Mohammad Alizadeh, and Sachin Katti. 2016. Cliffhanger: Scaling performance cliffs in web memory caches. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 379–392.

[8] Mostafa Dehghan, Laurent Massoulié, Don Towsley, Daniel Sadoc Menasché, and Y. C. Tay. 2019. A Utility Optimization Approach to Network Cache Design. *IEEE/ACM Transactions on Networking* 27, 3 (2019), 1013–1027. https://doi.org/10.1109/TNET.2019.2913677

[9] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. 2011. Understanding the impact of video quality on user engagement. *ACM SIGCOMM computer communication review* 41, 4 (2011), 362–373.

[10] Christine Fricker, Philippe Robert, and James Roberts. 2012. A versatile and accurate approximation for LRU cache performance. In *2012 24th international teletraffic congress (ITC 24)*. IEEE, 1–8.

[11] Google Cloud. 2026. Media CDN Origin Shield. https://cloud.google.com/media-cdn/docs/origins Google Cloud Media CDN Documentation. Accessed: 16 July 2026.

[12] Paul J Goulart and Yuwen Chen. 2026. Clarabel: An interior-point solver for conic programs with quadratic objectives. *Mathematical Programming Computation* (2026), 1–83.

[13] Frank P Kelly, Aman K Maulloo, and David Kim Hong Tan. 1998. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society* 49, 3 (1998), 237–252.

[14] S Shunmuga Krishnan and Ramesh K Sitaraman. 2012. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. In *Proceedings of the 2012 Internet Measurement Conference*. 211–224.

[15] Tian Lan, David Kao, Mung Chiang, and Ashutosh Sabharwal. 2010. *An axiomatic theory of fairness in network resource allocation*. IEEE.

[16] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Alexandros Moorthy, and Megha Manohara. 2016. Toward A Practical Perceptual Video Quality Metric. In *Netflix TechBlog / PCS*.

[17] Steven H Low. 2003. A duality model of TCP and queue management algorithms. *IEEE/ACM Transactions On Networking* 11, 4 (2003), 525–536.

[18] Steven H Low and David E Lapsley. 1999. Optimization flow control. I. Basic algorithm and convergence. *IEEE/ACM Transactions on networking* 7, 6 (1999), 861–874.

[19] Nimrod Megiddo and Dharmendra S Modha. 2003. {ARC}: A {Self-Tuning}, low overhead replacement cache. In *2nd USENIX Conference on File and Storage Technologies (FAST 03)*.

[20] Jeonghoon Mo and Jean Walrand. 2000. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on networking* 8, 5 (2000), 556–567.

[21] Ricky KP Mok, Edmond WW Chan, and Rocky KC Chang. 2011. Measuring the quality of experience of HTTP video streaming. In *12th IFIP/IEEE international symposium on integrated network management (IM 2011) and workshops*. IEEE, 485–492.

[22] Netflix. 2026. Netflix Open Connect. https://openconnect.netflix.com/en/. Accessed: July 2026.

[23] Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun. 2010. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review* 44, 3 (2010), 2–19.

[24] Stefan Podlipnig and Laszlo Böszörmenyi. 2003. A survey of web cache replacement strategies. *ACM Computing Surveys (CSUR)* 35, 4 (2003), 374–398.

[25] Zhenyu Song, Kevin Chen, Nikhil Sarda, Deniz Altınbüken, Eugene Brevdo, Jimmy Coleman, Xiao Ju, Pawel Jurczyk, Richard Schooler, and Ramki Gummadi. 2023. {HALP}: Heuristic aided learned preference eviction policy for {YouTube} content delivery network. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1149–1163.

[26] Rayadurgam Srikant and Tamer Başar. 2004. *The mathematics of Internet congestion control*. Springer.

[27] Srikanth Sundaresan, Walter De Donato, Nick Feamster, Renata Teixeira, Sam Crawford, and Antonio Pescapè. 2011. Broadband internet performance: a view from the gateway. *ACM SIGCOMM computer communication review* 41, 4 (2011), 134–145.

[28] Kevin Vermeulen, Loqman Salamatian, Sang Hoon Kim, Matt Calder, and Ethan Katz-Bassett. 2023. The central problem with distributed content: Common cdn deployments centralize traffic in a risky way. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*. 70–78.