Tero: Offloading CDN Traffic to Massively Distributed Devices

Juan Vanerio
University of Vienna

Lily Hügerich
TU Berlin

Stefan Schmid
TU Berlin & Fraunhofer SIT &
University of Vienna

ABSTRACT

To provide high performance and cope with ever-increasing traffic demand, Content Delivery Network (CDN) providers have started considering the use of multi-tier architectures, including simple caching devices that can augment their server infrastructure, resulting in a massively distributed caching network. These caching devices are usually geographically distributed, although with limited storage space and bandwidth (e.g., set-top boxes), potentially alleviating the servers' load.

This paper initiates the joint resource allocation and routing problem underlying such networks while providing at least a minimum bandwidth for each request. We present *Tero*, a system that maximizes throughput in such scenarios and leverages popularity forecasting to adapt to demand changes quickly.

In *Tero*, the CDN's edge server decides whether to serve each request locally or redirect it to a specific caching device, maximizing overall system throughput by offloading traffic to the device caches. To adjust to the highly dynamic nature of the demand patterns, *Tero* performs frequent near-future content popularity predictions and makes allocation decisions every few minutes. We model the optimization problem under these constraints and derive optimality properties using a Lagrangian formulation from which we design heuristic algorithms.

We evaluate *Tero* on a synthetic and a real-world large CDN request sequences, on ablation studies, and by comparing with an upper performance bound. *Tero* can reduce the edge server's throughput and provide sufficient bandwidth to each request, outperforming the competing baselines by up to 44% while being close to the performance of the ideal upper bounds. Also, *Tero* takes allocation decisions orders of magnitude faster than solving the exact problem.

ACM Reference Format:
Juan Vanerio, Lily Hügerich, and Stefan Schmid. 2024. Tero: Offloading CDN Traffic to Massively Distributed Devices. In *25th International Conference on Distributed Computing and Networking (ICDCN '24), January 4–7, 2024, Chennai, India.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3631461.3631556

ACKNOWLEDGMENTS

Research partially funded by the German Research Foundation (DFG), grant 470029389 (FlexNets), 2021-2024 and the Vienna Science and Technology Fund (WWTF) project, Fast and Quantitative What-if Analysis for Dependable Communication Networks (WHATIF), grant 10.47379/ICT19045, 2020-2024.

ICDCN '24, January 4–7, 2024, Chennai, India
2024. ACM ISBN 979-8-4007-1673-7/24/01.
https://doi.org/10.1145/3631461.3631556

1 INTRODUCTION

The Internet's commercialization has led to the migration of content and various applications online, such as video streaming, news, and advertisements. In 2022, global Internet traffic surged by 23%. In particular, video usage constituted 65% of all Internet traffic [32], and this content is mostly served from CDNs.

The proximity between users and their requested content is pivotal in enabling low-latency content delivery. Content Delivery Networks (CDNs) are content-driven networks that strategically store various objects in proximity to users, ensuring high-performance video and web services [36]. A typical CDN structure is composed of a first layer of *origin servers* maintaining vast content catalogs encompassing a wide range of digital assets and a second layer of *edge servers* (or just *servers*) of large capacity that cache a substantial fraction of the whole catalog. The servers are deployed at points of presence (PoPs) connected to Internet Service Provider (ISP) networks, achieving proximity to users [20].

The CDN's routing mechanisms direct users' requests to the nearest server, providing swift access to the cached content. Only if the content is unavailable there it must be transferred from the origin servers. This approach minimizes the need for long-distance data transfer, reducing latency and overall completion time of the content delivery process [20]. Nevertheless, nearest server routing may run into bottlenecks on the path from users to the edge servers [11].

An emerging optimization opportunity being explored by major CDN providers, consists in deploying a third CDN layer with many small *caching devices* that can be placed in block cabinets or users' premises, e.g. set-top boxes and DVRs [11], for local caching of popular content. An implementation of this idea in the context of 5G and dense femtocells deployment is the usage of cache-equipped base-stations, an attractive alternative to offload the growing backhaul traffic in mobile networks [13]. A key benefit of the distributed device constellation is the ability to serve more traffic, and in general to alleviate the servers' load. Also, as the CDN has to pay the ISP for all transfers from the server, serving traffic from the distributed devices may lower data transfer costs as these leverage residential Internet services [11]. Users may be offered benefits for their engagement. However, the devices are usually heavily constrained in storage space and bandwidth, thus introducing complexity in how to operate the caching system.

An enabler for such an approach is the HTTP Redirect message, which allows the local server to redirect users to caching devices without further DNS lookups. The server is then responsible for request routing and logging, which makes it an ideal candidate to act as a central controller for orchestrating the constellation of devices. For instance, the controller could determine the optimal content of a caching device and instruct it to update itself accordingly.

In this context, two natural objectives for the CDN provider are to minimize traffic costs and to extend system throughput by offloading the egde server while providing satisfactory user experiences. Therefore, two main sub-problems must be addressed: first, decide which content object should be placed on which caching devices (*allocation* problem), and second, decide which requests should be directed to which devices (*routing* problem).

To address the allocation problem, it is beneficial to estimate the traffic demand required for each content object (*popularity prediction* sub-problem). The latter can be inferred from time-stamped request access records stored in the servers' log files. While some methods predict long-term evolution of popularities [24, 33], they are complex and expensive in terms of memory and processing. They also require a considerable amount of data to fit, which may not be available.

Nevertheless, long-term predictions are unnecessary as long as the predictions and allocation mechanisms are fast enough to be rerun every few minutes. Counting item requests on short intervals is essentially a band-limited process: the number of hits observed during the following time slot will not differ much from the previous ones and can therefore be inferred from the immediate past. This maps to a concept of smoothness in the demand, as it is typically driven mainly by humans at scales much longer than just a few minutes. A sharp and fast change in the number of requests would be expected when this assumption does not hold (e.g., for machine-generated traffic). In this case, predicting popularities based on the recent past also helps as it can adapt to the new pattern quickly. Given the dynamic and non-stationary nature of request processes, making frequent short-term traffic demand predictions is an interesting option.

For the routing problem, it is necessary to keep track of the requests being served by each device in order to avoid overwhelming them (*load estimation subproblem*). Implementing tight monitoring would introduce high communication and operational expenses while estimating it centrally may induce a significant time and memory footprint.

There are many more challenges to overcome. The routing and allocation problems are intertwined, as preventing excessive user concurrence drives the placement of multiple replicas of the same content along multiple devices. Also, the devices may be very diverse regarding their capabilities, requiring careful consideration. Similarly, there are different content items with their own size and non-stationary demand patterns [12], and the past request sequences required to forecast said patterns may be short-lived and provide just timestamps.

Our contributions. This paper proposes *Tero*, a fast and centralized control content delivery system for massively distributed multi-tier cache networks comprising devices heavily constrained in both storage space and bandwidth capabilities. To the best of our knowledge, we are the first to address content delivery while aiming at reducing costs and maximizing throughput by offloading a large amount of traffic to such configuration of caching devices while guaranteeing a satisfactory user experience under dynamic demand patterns and diverse content sizes. We provide a formal model of the CDN, and by careful analysis based on a Lagrangian formulation, we characterize the properties of the optimal solution.

Tero includes a novel allocation mechanism leveraging short-term content popularity prediction, which relies on recent past request process history readily available in log files of CDN servers. We compared our simple prediction mechanism based on short time intervals (a few minutes) against BHT-ARIMA [33], a state-of-the-art approach for multi-valued time series forecasting, and consistently outperformed it on the available datasets. The allocation mechanism is a greedy heuristic algorithm developed upon the optimal properties found in the model's analysis. For routing, *Tero* forwards requests online to those devices with the lowest estimated relative concurrency.

We evaluate *Tero* empirically based on both synthetic and real-world content request data by systematically replacing different system components with alternatives. The latter dataset contains a timestamped request sequence spanning over 18 hours of traffic and involving more than a million content objects. As a reference, in our measurements, allocations for a million content objects and tens of thousands of devices take less than two minutes to be computed in a single thread on a mildly provisioned virtual server. We compared *Tero* against baselines constructed by replacing a component with a simpler alternative (baseline) and evaluated according to the 95-percentile (over time) of the throughput of the edge server, the Byte-Hit-ratio (BHR), and the relative concurrencies on caching devices. For instance, popularities are compared against true popularities when available, a baseline for content placement is getting replicas proportionally to the item popularity, and for concurrencies we compare against a random-router that performs optimally for load-balancing purposes but pays no attention to each request offered bandwidth. *Tero* performs better than the feasible baselines we compared against. Also, *Tero* keeps a small performance gap with respect to a natural ideal performance upper bound. In concurrency, we found improvements of up to 11 times. In server throughput, we achieved a throughput reduction of up to 44% on the synthetic trace and 23% on the real-world trace.

The remainder of this paper is organized as follows. Section 2 describes the problem as an optimization model and analyzes it. Section 3 presents *Tero* system's design, and Section 4 provides detailed evaluation of its performance. Finally, we review the related work in Section 5 and provide the conclusions in Section 6. Technical details can be found in the Appendix.

2 MODEL AND OPTIMIZATION

2.1 Model overview

We consider a multi-tier CDN delivering content to its users from high-tier servers or ideally from caching devices installed in their homes or neighborhood, making up the lowest CDN tier. These devices could be set-top boxes, network-attached storage devices, or other in-network devices [11, 13]. Although these devices are limited in storage and bandwidth, they extend the overall system capacity to the area covered by a CDN edge server.

To formalize the problem, consider that each content object or *item* in the catalog is indexed by $i \in [1, M]$, has size s_i (in bytes), and popularity p_i (its expected number of requests per time unit).

Items can be hosted in *caching devices* $d \in [1, D]$ and in the server ($d = 0$). Each device d is characterized by its bandwidth B_d (used to serve content to users) and storage capacity C_d. The number of concurrent requests is indicated by r_d.

The model considers a single geographical area including the lowest CDN tier (caching devices) and an edge server. The latter hosts the whole item catalog $[1, M]$ and has unlimited bandwidth.

The main objective is to collectively serve traffic H corresponding to the request sequence σ using as much traffic from the devices as possible. Offloading the server traffic allows more content to be served with less usage of the ISP's backbone, which may reflect a smaller cost per transferred byte. The cost of the involved requests and messages is deemed negligible.

Matrix $X = \{x_{id}\}$ of binary control variables indicates whether item i is cached in d ($x_{id} = 1$) or not ($x_{id} = 0$). Each device may transfer data up to its bandwidth, and the accumulated size of its cached items may never exceed its storage capacity. To fetch a new item, a device downloads it from the server at finite (download) bandwidth b_d and may evict existing content to make enough space.

We model set of users as a single source of requests with no preference for any caching device: they belong to the same geographical area and experience similar delays from any device or server. The fraction of traffic provided to the users from device d for item i is captured by the routing matrix $A = \{\alpha_{id}\}$.

Finally, for a satisfactory user experience every request should enjoy at least bandwidth δ regardless of where it is served from. The traffic served from the devices follows a processor-sharing discipline, roughly mapping to having the device's bandwidth B_d split along concurrent requests. Thus, the number of concurrent requests should not exceed $R_d = \frac{B_d}{\delta}$. Table 2 (included in Appendix A.1) summarizes the notation used in the problem.

2.2 System model

The objective of minimizing the traffic H_s being delivered from the server while serving sequence σ can be stated as the following optimization problem:

$$\min_{X,A} \quad H_s = \sum_{i=1}^{M} s_i N_{i0} \tag{1}$$

$$\text{subject to:} \quad \sum_{d=0}^{D} \alpha_{id} = 1 \quad \forall i \in [1, M] \tag{2}$$

$$\sum_{i=1}^{M} x_{id} s_i \leq C_d \, \forall d \in [1, D] \tag{3}$$

$$\sum_{i=1}^{M} s_i N_{id} \leq T B_d \, \forall d \in [1, D] \tag{4}$$

$$r_d \leq R_d = \frac{B_d}{\delta} \, \forall d \in [1, D] \tag{5}$$

$$0 \leq \alpha_{id} \leq x_{id} \leq 1, \forall i \in [1, M], d \in [1, D] \tag{6}$$

$$x_{id} \in \{0, 1\} \forall i \in [1, M], d \in [1, D], d > 0 \tag{7}$$

where N_{id} be the number of requests to item i served from device d and T the elapsed time.

We want to find the minimizers X^* and A^* that indicate optimal allocation and routing parameters for the objective function (1), which directly represents the number of bytes delivered from the edge server. Constraint (2) represents demand conservation, as fractions of traffic not served from any caching device are delivered from the server. Constraint (3) and constraint (4) state that devices must not exceed their storage capacity and their bandwidth, respectively. Constraint (5) expresses the satisfactory user experience condition by which all requests should enjoy at least transfer rate δ. Finally, constraints (6) and (7) consider the valid value ranges for the control variables.

2.3 Decomposing the problem and routing

Attempting to solve instances of this problem using off-the-shelf solvers typically results in excessively long (hours or days) execution times, rendering the solution unusable in real-world deployments. Instead, **we decompose the problem into smaller subproblems, identify properties of the optimal solution and use them to build heuristic algorithms**.

First, we provide a solution for the routing with given allocations. Note that the total traffic demand H can be split into:

- Traffic H_c served from the devices,
- Traffic $H_s^{(u)}$ requesting unallocated items, flowing from the server to the users,
- *Excess traffic* $H_s^{(e)}$ from the server due to devices hosting replicas of the requested item being maxed out of concurrent requests.

Then, for any given allocation X, the optimal routing A is the one that *minimizes excess traffic* $H_s^{(e)}$. This stems from the fact that demand is conserved and that if the allocation is given, then so is the traffic from unallocated items $H_s^{(u)}$. As excess traffic can only be generated from devices operating at maximum concurrency, the optimal routing policy π can be achieved by forwarding new requests to the devices with the smaller relative concurrency r_d/R_d.

2.4 Allocation model

To address the allocation problem, we focus on short time interval of length T during which the item popularities and the request process do not observe large variations, i.e., the requests counting signal is band-limited. This assumption holds when the number of requests for an item on a time interval is similar to the previous ones. This is to be expected of human-driven traffic, whose popularities typically range in hours [2, 38]. It suffices then to consider time intervals spanning a few minutes.

Let λ be the average request process intensity during the time interval. We then replace the number of requests N_{id} served for item i from device d on equation (1) with $T\lambda p_i \alpha_{id}$ and remove constraints concerning only the routing to obtain an offline model:

$$\max_{\mathbf{x}} \quad H(\mathbf{x}|\pi) = \sum_{i=1}^{M} \lambda s_i p_i (1 - \alpha_{i0}|\pi) \tag{8}$$

$$\text{subject to:} \quad \sum_{i=1}^{M} x_{id} s_i \le C_d \, \forall d \in [1, D] \tag{9}$$

$$\lambda \sum_{i=1}^{M} s_i p_i \alpha_{id} \le B_d \, \forall d \in [1, D] \tag{10}$$

$$\sum_{d=0}^{D} \alpha_{id} = 1 \quad \forall i \in [1, M] \tag{11}$$

$$0 \le \alpha_{id} \le x_{id} \le 1, \forall i \in [1, M], d \in [1, D] \tag{12}$$

In *Tero*, we use a Lagrangian formulation, which has already been used in related optimization problems [3, 10, 23, 35]. According to [8], the Lagrangian method is an efficient approach for solving even large-scale instances. The fundamental concept underlying the Lagrangian method involves reframing the constrained optimization problem as an unconstrained one. The Lagrangian for model 2.4 can be expressed as follows after introducing dual variables $\{\gamma_d\}$, $\{\mu_d\}$ and $\{\theta_{id}\}$, $i \in [1, M]$, $d \in [1, D]$:

$$\mathcal{L} = -\sum_{i=1}^{M} s_i p_i \alpha_{i0} + \sum_{d=1}^{D} \gamma_d \left[\sum_{i=1}^{M} s_i x_{id} - C_d \right] + \sum_{d=1}^{D} \mu_d \left[\sum_{i=1}^{M} s_i p_i \alpha_{id} - \frac{B_d}{\lambda} \right] + \sum_{d=1}^{D} \sum_{i=1}^{M} \theta_{id} \alpha_{id} (\alpha_{id} - x_{id}) \tag{13}$$

We explore the Karush-Kuhn-Tucker (KKT) optimality conditions to identify desirable properties in the solution. Then, the slacking conditions for the bounds on the valid range of A satisfy:

$$\theta_{id} \alpha_{id} (\alpha_{id} - x_{id}) = 0 \quad \forall i \in [1, M], \forall d \in [1, D] \tag{14}$$

and the stationary condition can be expressed as:

$$0 = \frac{\partial \mathcal{L}}{\partial \alpha_{id}} = (\mu_d - 1) s_i p_i + \theta_{id} (2\alpha_{id} - x_{id}) \tag{15}$$

where we replaced the demand constraint (11) into the first term of the right-hand side. Now if we multiply both sides of the stationary condition by $\alpha_{id} (\alpha_{id} - x_{id})$, and recall equation (14) and the fact that $s_i p_i > 0$ we get:

$$(1 - \mu_d) \alpha_{id} (\alpha_{id} - x_{id}) = 0 \tag{16}$$

The slack for the bandwidth constraints satisfy:

$$\mu_d \left[\sum_{i=1}^{M} s_i p_i \alpha_{id} - \frac{B_d}{\lambda} \right] = 0 \, \forall d \in [1, D], d > 0 \tag{17}$$

from which we know that if device d has remaining bandwidth on the optimal solution, then $\mu_d = 0$, and after equation 16, $\alpha_{id} (\alpha_{id} - x_{id}) = 0$. Otherwise, d has no remaining bandwidth and then $B_d - \lambda \sum_{i=1}^{M} s_i p_i \alpha_{id} = 0$. Therefore, the following condition holds in the optimal solution:

$$\left(B_d - \lambda \sum_{i=1}^{M} s_i p_i \alpha_{id} \right) \alpha_{id} (\alpha_{id} - x_{id}) = 0 \tag{18}$$

Optimality conditions. Two *initial optimality conditions* can be inferred from the original model (Section 2.2). First, in the optimal allocation, no device can cache an additional object (either an unallocated item or a replica) without exceeding its storage capacity or bandwidth. Second, swapping items between devices, or dropping a cached item (or a replica thereof) to cache an unallocated item, cannot improve the transfer rate.

Third, from equation (18) we can draw the following conclusions for the optimal solution:

- Combining with the initial optimality conditions, if some item i has replicas (i.e., more than one allocated instance), then it must have fractional routing ($0 < \alpha_{id} < 1$). Otherwise, we could swap the unused replica with another item.
  - The number of replicas must be kept to a minimum: the more there are, the more storage space is needed without producing more demand.
- If some device d has remaining bandwidth available, then every item i allocated in d has an integer routing factor $\alpha_{id} \in \{0, 1\}$.
  - Due to equation 2 and the initial optimality conditions, it follows that i has no replicas.
- All devices holding a replica of an object must be operating at full bandwidth.

Fourth, the optimal solution must keep oversubscription (traffic demand exceeding the device's bandwidth) as low as possible to minimize excess traffic. Otherwise, dropping an item to make space to spawn a replica of a highly-demanded object would provide more throughput, which would contradict optimality. From these considerations, if each device has allocations as close as possible to *simoultaneously* fill its bandwidth and storage space, the system's throughput would be at its maximum.

2.5 Single Device Model

A natural performance upper bound for *Tero* is the Single Device Model (SDM), composed by a single caching device whose bandwidth is the accumulated bandwidths of all devices on the original model: $B_{SDM} = \sum_{d=1}^{D} B_d$ and the same goes for the storage space ($C_{SDM} = \sum_{d=1}^{D} C_d$). As the SDM does not benefit from having item replicas, its storage space is used more efficiently and may output a larger throughput due to additional cached items. Also, the allocation of the SDM is simply based on popularities (see appendix A.4 for details) and there is no routing problem in the SDM.

There are two cases in which SDM and the original model perform similarly: when the demand is small with respect to the total system bandwidth and when it is too large. In the former case, items can be cached on any device without maxing out their bandwidth, avoiding replicas. In the latter, the traffic demand is high enough to have all devices work at full bandwidth.

3 THE TERO SYSTEM

Tero is designed with one module for each task: popularity prediction, allocation, routing, and concurrency estimation. A central controller orchestrates the modules' interactions. An overview of *Tero* is shown in Figure 1. In the following paragraphs and Sections, we describe each component.

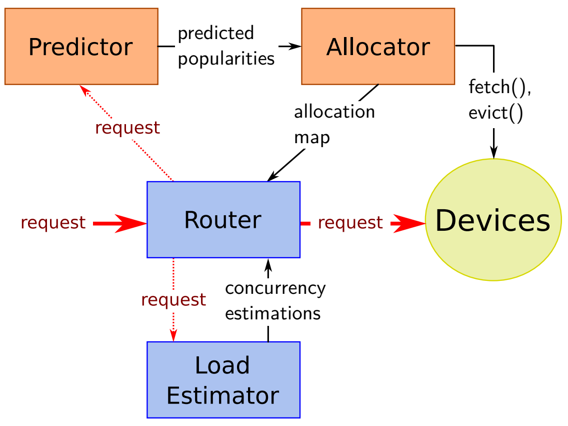

Figure 1: *Tero* design overview.

Short-time intervals (*time slots*) during which item popularity and request intensity are assumed to be roughly constant are considered (see Section 2.4 for details). At the beginning of each time slot, the controller invokes the predictor component to estimate the items' intensities based on historical data. The allocator module uses the predictions to compute a new *allocation map*. With the updated allocation map, the controller instructs each caching device which new content to fetch and which to evict.

On the other hand, the router and the concurrency estimator module (CEM) operate online. Upon a new request, the router identifies a set of devices holding a replica of the requested object using the allocation map. Then, it queries the CEM for their estimated relative concurrences and selects the least loaded one. A redirect message is sent to the user to refer her toward the selected device. If the requested item is not cached or no device caching the item can serve an additional request, then the item is transferred from the server.

There is an online mechanism to cope with prediction errors on-the-fly. It detects any item transferred from the server too many times and triggers an on-demand allocation request for the related item into a device with a low load.

3.1 Popularity prediction

The problem. To place the content, *Tero* needs accurate predictions of the items' popularity and request intensities during the next time slot. The available historical data only contains each access request timestamp and item id. From the latter, the item size can be obtained. There is no other available side information source delta and exponential decay-type features [40] could be built from the timestamp information, augmenting the data. There is though a trade-off between holding richer data about fewer requested content items, or simpler data on more items.

Also, the available dataset may span just a few hours, making it impossible to learn any seasonality in the data, e.g., naturally occurring daily, weekly, or monthly patterns. Given the small amount of data available, it would be challenging to train deep learning models. Even more, the full catalog is unknown and may be dynamic, leaving historical log-based data as the only source of catalog discovery.

Algorithm 1 Offline Allocation.

```
1:  Initialize $v_i \leftarrow \lambda p_i s_i\ \forall i$, $b_d \leftarrow B_d$, $c_d \leftarrow C_d\ \forall d$,
2:  while possible do
3:      $j \leftarrow \arg\max \{v\}$
4:      Let $D' = \{d' : c_{d'} \geq s_j \wedge j \notin d' \wedge b_{d'} > 0\}$
5:      Allocate $j$ in $d^* = \arg\max_{d' \in D'} \frac{b_{d'}}{c_{d'}}$.
6:      $c_{d^*} \leftarrow c_{d^*} - s_j$
7:      if $b_{d^*} \geq v_j$ then
8:          $b_{d^*} \leftarrow b_{d^*} - v_j$
9:          $v_j \leftarrow 0$
10:     else
11:         $b_d \leftarrow 0$
12:         $v_j \leftarrow v_j - b_{d^*}$
```

The solution. We propose a simple approach capable of providing speed and consistency to the predictions for as many popular items as possible on each time slot. Start by creating a multi-valued time series by counting the requests to each item seen more than once during short time intervals of fixed length. Ignoring single-hit items saves a significant caching space as most items are requested just once [20]. Then, we use *moving averages* of the most recent x time intervals as the next prediction. The justification is that if the involved time intervals are short enough, then the request process can be deemed as stationary for their duration, and in such conditions the moving average is a good estimator of the true intensity.

3.2 Allocation

The problem. We need to decide on item allocations based on their popularity predictions, sizes and the previous allocation decisions (*brownfield scenario*). Decisions must be made fast enough to handle tens of thousands of devices and millions of items without exceeding the duration of a time slot.

Knowing the expected demands, the problem can be formulated as a linear integer problem and solved with an off-the-shelf solver. Nevertheless, that computation would take a prohibitive amount of time for the high number of items and devices.

The solution. We propose algorithm 1; a deterministic, greedy, and fast algorithm to allocate items to devices for each time slot. The algorithm considers the item's requested demand and size in an attempt to try to simultaneously fill the devices' bandwidth and storage space, as this concept captures the properties of the optimum as discussed in Section 2.4.

First, we initialize variables v_i with the expected unallocated demand for item i, the remaining device d's bandwidth b_d and its remaining storage space c_d. In (line 3) the algorithm picks the next item j with the largest unallocated demand and (line 4) identifies a set D' of candidate devices for allocating it. The members of D' have enough available bandwidth and storage space and still have not cached j. Then, (line 5) chooses the device d^* in D' with the highest ratio of remaining bandwidth over remaining storage space as the best candidate. As discussed in Section 2.4, the ideal operational point for the device is to be simultaneously at full bandwidth, space depleted and with no oversubscription. *Tero* places high-demand items on devices with the largest ratio to aim at serving more demand while minimizing the ratio, which in turn fosters distributing elements along other devices. The remaining space of d^* is updated in (line 6). If there is enough bandwidth available in the device after the allocation, the algorithm updates it by subtracting the item's demand (line 8) and sets in zero the remaining traffic demand for j (line 9). Otherwise, the item's demand is updated by subtracting the last remaining bandwidth of the device (line 12), and the device is declared as bandwidth-depleted (line 11). This last step allows the item to spawn replicas to satisfy the demand.

Algorithm 1 provides a solution for an empty fleet of devices. In practice, devices will already have cached items from previous time slots. To cope with such scenarios, we introduce an additional *stickiness* mechanism: once an item (or replica) is cached in a device, the allocator remembers the placement and avoids executing the item's corresponding algorithm's iteration in the next time interval. The past allocation is cached until the replica or item is evicted from a device. The stickiness scheme avoids reallocations, which in turn minimizes fetching traffic from the server to the devices.

3.3 Routing

The problem. The allocation-aware router module is responsible for deciding which device to forward a new request without introducing a significant delay. Requests must be served by the edge server if the item is not cached anywhere (unallocated) or if the CEM warns that all devices containing an item's replica are serving their maximum number of concurrent requests (overload).

The solution. According to the analysis in Section 2.3, the router should redirect the request towards the device with the lowest current relative concurrency among those holding an item's replica. As the concurrency is not directly observable, the router queries each device's CEM *estimated makespan* as a proxy measurement and forwards the request to one with the smaller value.

Finally, the router module also provides online resiliency to prediction errors. For this, it keeps track of the times each item is delivered from the server using a variant of Least Recently Used (LRU) equipped with a hit counter per item (A *counting-LRU*). A *Cache-on-second-hit rule*[20] is implemented to avoid flooding the LRU cache. As popular objects should have been cached on the devices and served from there, a prediction error is assumed if the LRU-counter value exceeds a predefined threshold. *Tero* subsequently removes item i from the counting-LRU and instructs the device with the lowest makespan to fetch a replica. If additional space is required, the chosen device evicts local content according to a LRU-k rule [25]. This way, any unforeseen burst in popularity can be corrected on-the-fly.

3.4 Concurrency estimation

The problem. A satisfactory user experience depends directly on each request having sufficient bandwidth, i.e., at least δ. This requirement maps to avoid serving too many requests simultaneously from a caching device. As it would be comunicationally expensive to inquire the devices' load upon each new content request, the router needs a local (indirect) method to estimate the number of concurrent requests on each device.

Algorithm 2 Concurrency Estimation Module: Makespan estimation

```
1:  Inputs: inertia parameter $\gamma$, minimum bandwidth $\delta$.
2:  **Initialize** $ts_d \leftarrow 0, \hat{s}_d \leftarrow 0, m_d \leftarrow 0 \forall d > 0$,
3:  **procedure** UPDATE_BY_TIME(device $d$, time $t$)
4:      $m_d \leftarrow \max(0, m_d - t + t_s)$
5:      $ts_d \leftarrow t$
6:  **procedure** UPDATE_UPON_REQUEST(device $d$, time $t$, item $i$)
7:      UPDATE_BY_TIME$(d, t)$
8:      $\hat{s}_d \leftarrow \gamma \hat{s}_d + (1 - \gamma) s_i$
9:      $m_d \leftarrow m_d + \frac{s_i}{B_d}$
10: On each incoming request $(t, i)$ to item $i$ at time $t$:
    UPDATE_UPON_REQUEST$(d, t, i)$
11: When CEM is queried about device $d$ at time $t$:
12: UPDATE_BY_TIME$(d, t)$
13: **if** $m_d > \frac{\hat{s}_d}{2\delta}$ **then**
14:     return "Device $d$ is overloaded."
15: **else**
16:     return $m_d$.
```

The solution. *Tero* computes a proxy metric: the estimated makespan (m) or time until completion of each device. This computation requires a timestamp and two scalar values per device, and updates are made on demand, resulting in a small footprint. Algorithm 2 describes its working.

The algorithm has two input parameters: minimum bandwidth δ and tracking inertia γ (line 1), and defines three internal parameters for each device; its estimated makespan m_d, its estimated average hosted items size $\hat{s}_d$ and a timestamp ts_d. There are two main procedures: "UPDATE_BY_TIME" (line 3) that decreases the makespan estimation reflecting the elapsed time, and "UPDATE_UPON_REQUEST" (line 6), which is called to update average size tracker $\hat{s}_d$ (line 8) and the makespan with the expected time to serve the new request at full bandwidth (line 9).

Every time the CEM finds a makespan m_d above the threshold $\frac{\hat{s}_d}{2\delta}$, the corresponding device is deemed overloaded by concurrent requests and removed from the routing process (line 14). Otherwise, its makespan m_d is returned to the router (line 16). The threshold derivation is presented in Section A.2.

4 EVALUATION

In this section, we conduct experiments to evaluate the performance of *Tero* on two traffic traces. The first is a synthetic trace consisting of the superposition of multiple Poisson request processes, a well-understood process family to test caching systems. The second is a dataset extracted from real-world log files of a CDN server spanning 18 hours (the *log-based* dataset).

Methodologically, we first perform an experiment to assess the validity of *Tero*'s prediction method. Then, for each traffic trace, we first perform an ablation study on each of *Tero*'s components and compare the performance against an upper bound (the SDM).

We implement an event-driven simulator that captures the request processes with variable interarrival times and item sizes. It implements the server, the devices, and every component described in section 3. For each ongoing request, it thoroughly keeps track of the content download progress and status. Each device efficiently implements a processor-sharing discipline to share the bandwidth among concurrent requests on each device. The simulator is used on the two datasets and keeps track of all relevant statistics, including the throughput on the server, which also accounts for the traffic due to item fetching by devices.

4.1 Settings

All experiments were executed using a single instance of a 64 GB RAM virtual server running Ubuntu 22.04.2 LTS, Python 3.8. On all the simulations, we defined a setup with a single CDN server and two sets of caching devices called *fleets*. The first fleet is composed of 7000 devices with 50 Mbps of bandwidth and 32 GB of storage space, and the second fleet has 14000 devices of 20 Mbps and 32 GB. The minimum request bandwidth was fixed at $\delta = 1$ Mbps. This setup closely mimics the one of the CDN that provided the log files.

For the predictions, unless otherwise stated, we counted requests in 15 s time windows to create the time series and we used the last 300 s moving average to predict item popularity. For the synthetic source, we defined a catalog of $M = 100000$ items whose popularities follow a Zipf distribution. Item sizes range between 100 KB and 1 GB (average: 7 MB) and the request process intensity is set to $\lambda = 10$ kreq/s, matches the values of the log-based dataset. We performed 12 independent simulations of 600 s (simulated time) each and reported their results.

For the log-based experiments, we use the same setup as with the synthetic source, although intensity, popularity distribution, and numbers of cacheable items are implicit. We simulated different time intervals during the day: 30 minutes (19:30-20), 15 minutes (16:00-16:15), and 10-minute (8:00-8:10) intervals of log traffic from the dataset. The *warmup time*, i.e., the time before gathering statistics, was set at 120 s.

4.2 Content and popularity prediction

We compare the performance of the following popularity prediction techniques in estimating the true item popularity of the log-based dataset request process.

- **Simple_x**: per item incidence moving average on the last x time slots, the method used by *Tero*.
- **Linear**: Linear extrapolation of the last 10 time slots used as the next prediction.
- **BHT-ARIMA** [33]: A **state-of-the-art** method for high-quality multivalued time series prediction. Performs a sophisticated embedding of the series into a lower-dimensional space and predicts using ARIMA.

Figure 2 shows the results for predicting the top 50 most popular items using the different methods after 6 predictions and a window of six time steps. It can be seen that the "simple" methods are consistently more accurate than the other baselines, justifying our decision to use them as fast predictors for the short term.

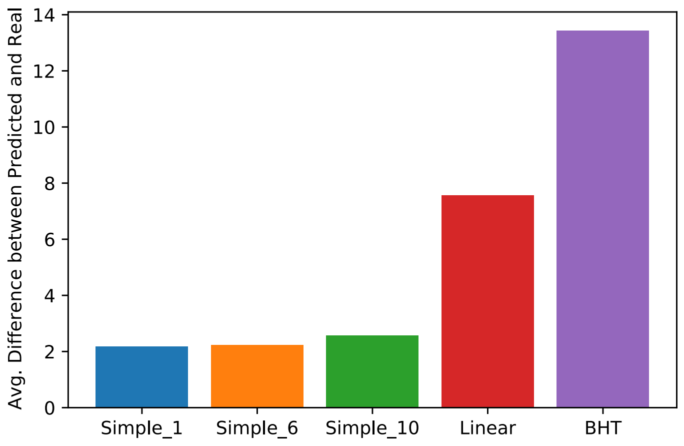

Figure 2: Average absolute error for different popularity prediction schemes on identifying top 50 most popular objects on the next time slot.

Additional indirect assessment of this method is subsequently performed on experiments comparing against baselines leveraging the true popularity of synthetic traces (see Section 4.3.4).

4.3 Poisson source evaluations

4.3.1 Allocator. We compare against an identical system in which our allocator module is replaced with another module from:

- Allocator baseline: Instantiates replicas proportionally to the item's popularity.
- Allocate by popularity: Follows algorithm 1 but considers items popularity alone instead of their traffic demand.

The latter is meant to be a very competitive variation that may achieve as good results as *Tero*'s. In particular, allocating by popularity is very good when the optimum has few replicas (e.g., SDM) or the demand is overwhelmingly intense for the system's bandwidth.

Figure 3 shows the 95-percentile (over time) throughput of the CDN server. This percentile of traffic is chosen as ISPs usually use it to charge customers. Each box extends from the first to the third quartile of the experiment data, with a line at the median (highlighted as a narrow waist on the box). The whiskers indicate to the farthest data point within one and a half times the inter-quartile range from the box boundaries.

In this case, the goal is to achieve low values. For the same request sequences, the allocator achieves (in median) 44% less traffic than the baseline and 20% less traffic than the popularity-based method. It also experiences lower variance, reflecting a stable operation.

We also studied the Byte-Hit-ratio (BHR), i.e., the percentage of bytes transferred from the caching devices over the full traffic demand. For the same request pattern, a higher BHR indicates a better performing cache system. In median, the proposed Allocator outperforms the alternatives by 1.39x (Allocate by Popularity) and 9.47x (Allocator Baseline).

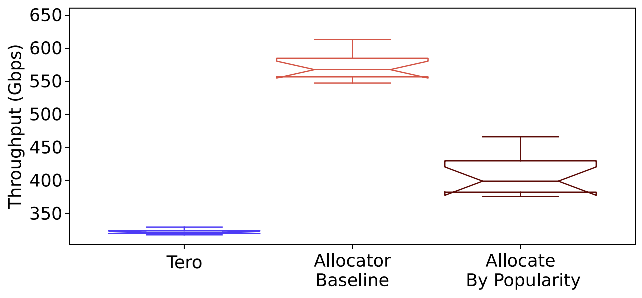

Figure 3: Edge Server 95-percentile Throughput with synthetic source.

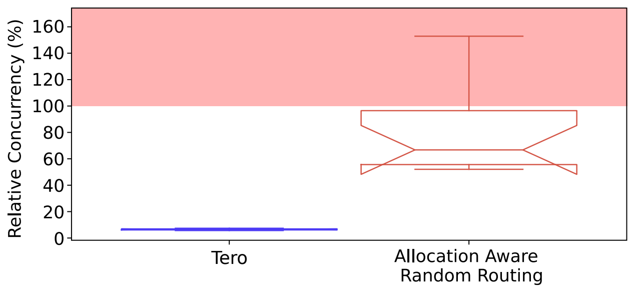

Figure 4: Relative Concurrency on caching devices for routers with synthetic source.

4.3.2 Router. We compare the proposed allocation-aware minimum-makespan router module against a *random router* that chooses randomly between the devices holding a replica of the requested item. In median, *Tero*'s router gets slightly higher (2 %) server throughput than baseline. This is to be expected, as the random router is optimal to balance the traffic *without considering the concurrency constraints*, that is, the user experience for each request.

Figure 4 shows the 95-percentile (over time) of the concurrency (on average over all devices), for both types of routers. The upper zone in red shows the region that incurs a constraint violation by overloading the caching devices. It can be seen that the makespan-based router has approximately 11 times less occupancy than the random router, between 0% and 10%, which results in a satisfactory experience for the user.

Additional delay considerations. The router is an online component, so it must be fast enough to avoid becoming a bottleneck. We compare the additional delay introduced by the makespan-based router on three different request process intensities: 6 kreq/s, 12 kreq/s, and 24 kreq/s, which are below, similar, and double the system bandwidth, respectively. We found that the router can route in real time (below requests' interarrival time, RIT) for the first two, even with a single thread. The last scenario required two threads to route below RIT. In all cases, the delays introduced are below 1 ms per request. The processing time increments for the router are sublinear, which enables scalability. Further scalability can be achieved via straightforward parallelization of the module.

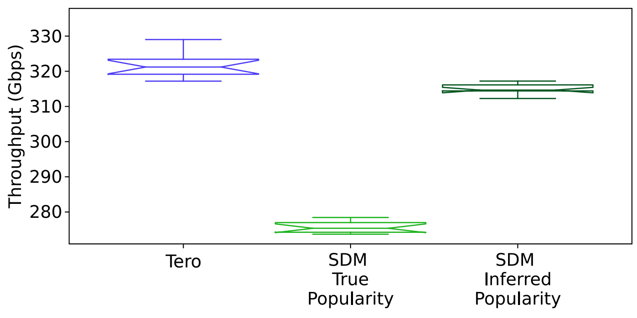

Figure 5: Edge Server 95-percentil Throughput against SDM with synthetic source.

4.3.3 Concurrency Estimator. We now evaluate the performance of CEM against the true concurrency on the devices, which is too expensive to use in an actual massive deployment. We compare in terms of the system's achieved throughput when using the estimated or the real value. We found a difference (in median) below 1% than the traffic achieved using the true concurrency directly, a result validating *Tero*'c CEM design.

4.3.4 Single Device Model. We now proceed to a full-system-wide comparison. For this purpose, we evaluate the performance of *Tero* against the Single Device Model (SDM), a natural bound for *Tero*'s performance. For completeness, we use two versions of the SDM; an idealized one that uses the true item popularities and another that uses predicted popularities and item discovery as *Tero* does.

It is important to recall that given its characteristics, the SDM using actual popularity information can approximate arbitrarily close to the optimum performance. When the SDM uses predictions instead, *Tero* performs very competitively; just 2% more traffic (figure 5) and 3% less BHR (in median). These results show that the performance of *Tero* is close to what can be achieved.

4.4 Log-Based Source Evaluations

To evaluate *Tero*'s performance on the real-world request sequence, we follow the same evaluation methodology as with the synthetic source whenever possible. As the log-based dataset has no fixed intensities and popularities to compare against, some baselines are no longer available.

4.4.1 Allocator. Figure 6 compares performance in terms of the server's throughput. There is a 23% traffic reduction for our proposed Allocator with respect to the baseline. In particular, *Tero* does slightly better (roughly 3%) than the popularity-based allocation method, reflecting that the latter is a good alternative in some scenarios.

4.4.2 Router. For the router, we observe the same pattern as for the synthetic source: the random router baseline achieves slightly less edge server throughput (again, as expected) but with a much larger average system occupancy: 48% higher in median (Figure 7). This translates to longer times for fetching content by the users and frequent violations of the required minimum bandwidth per request, resulting in a worse user experience.

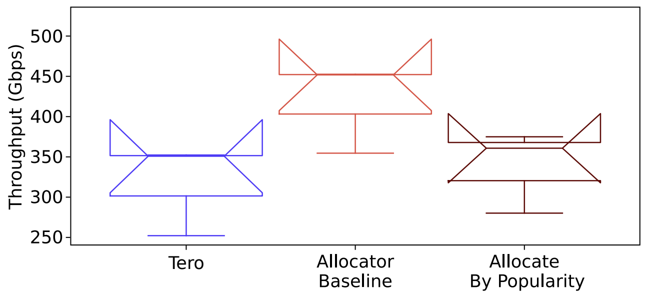

Figure 6: Log-based server throughput for Allocators for log-based source.

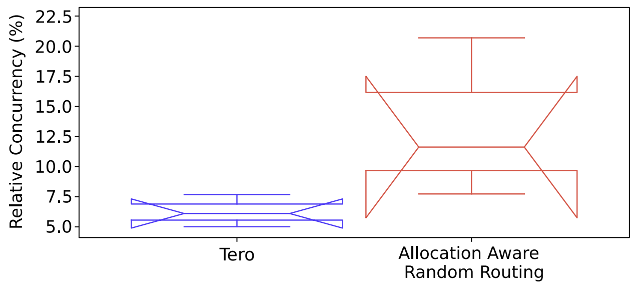

Figure 7: Relative Concurrency on caching devices for log-based source.

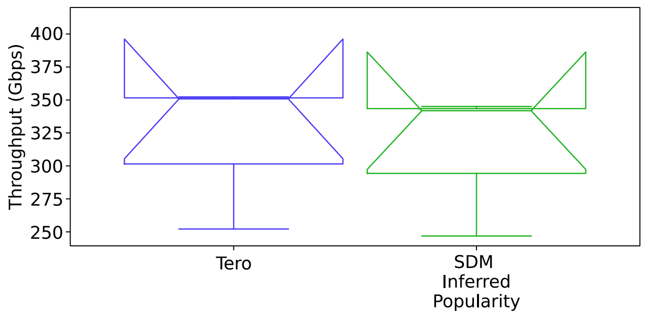

Figure 8: Throughput aganst SDM for log-based sources.

4.4.3 Concurrency Estimator. The results obtained for the Poisson-bases synthetic source are confirmed on the real-world log-based dataset: less than 1% difference in edge server's throughput. This shows that *Tero* can accurately estimate the devices' load and redirect requests appropriately.

4.4.4 Single Device Model. In this context, as the true popularity is unknown, we can only compare *Tero* against the inferred-popularity SDM. The server throughput is shown in Figure 8. As with the synthetic source, a performance gap of only 2% in server throughput is observed, showing that *Tero* is performing close to the SDM.

4.5 Results Summary

On a component-by-component ablation study on both request sources, every module of the *Tero* has outperformed the corresponding baselines on all relevant metrics. In the synthetic source experiments, the Allocator achieved up to a 44% traffic reduction with respect to baseline systems. The router achieved very similar throughput to the baseline but with one order of magnitude lower relative concurrency on the devices. A special note goes for the Concurrency Estimator Module, which for both sources performed had less than 1% difference against knowing the true concurrency. The router and the CEM make their estimations in real-time and could serve a more extensive deployment through parallelization.

Regarding system-wide performance, *Tero* exhibited comparable performance to SDM using inferred popularity. Also, allocation computations took under 10 s in the experiments with discovered items, which are usually a few thousand, and less than two minutes on the simulated one million items catalog. This is an encouraging result, given that solving the problem exactly takes orders of magnitude more time.

5 RELATED WORK

Content distribution and its underlying optimization problems have been studied intensively in the literature [20, 39, 42]. While classic caching solutions often revolve around Least Recently Used (LRU) and Least Frequently Used (LFU) strategies [18, 21], our focus is on a distributed network setting with many heterogeneous caches, leading to a different allocation and routing problem, as well as a scalability requirement.

Learning approaches have also been considered in caching systems before. A first type of works focuses on predicting content objects' popularity in order to prefetch the content [6, 24, 37, 41], while others optimize a predefined cache utility function through online learning [27, 31]. Popularity-estimation-based algorithms are more adaptive versions of traditional caching policies and tend to overlook temporal dynamics. A recent example is [24], which approaches popularity prediction as a sequence classification problem employing a deep-learning LSTM Encoder-Decoder model. Nevertheless, the method requires a considerable spanned time on the training dataset while assuming that items are independent. Finally, the method was tested only on roughly 1500 content items. The idea is similar to *Tero* popularity prediction, although the former attempts to predict sequences for long periods, while we focus on very short time scales instead, for which the band-limited hypothesis on the request counting sequences greatly simplifies the problem.

Online caching decision-making algorithms are often more complex (e.g., they usually need to deal with delayed rewards) and more sensitive to hyper-parameters [5, 17]. Several papers get inspiration from Belady's MIN algorithm [4] for eviction decisions, either by emulating it directly [15, 30] or using machine learning techniques like LFO [5] to imitate Belady's decisions. In contrast, LRB [34] predicts the next arrival time of the cached objects and then evicts content using the Belady approach on the estimations. LFO and LRB use manual feature engineering, limiting its applicability and generalizability.

Work	Popularity Prediction	Load Estimation	Routing	Allocation	Hetero-geneous devices	Hetero-geneous items' sizes	10K scalability	fast solving	per-request bandwidth support
[16, 28]			✓	✓	✓				
[12, 24, 34, 40]	✓					✓			
[9]			✓	✓	✓				✓
[13, 14]			✓	✓	✓		✓		✓
[29]			✓	✓	✓	✓			
[19]			✓	✓	✓		✓		
Tero (ours)	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: Related works characteristics.

To be simple and fast, *Tero* uses a popularity-estimation method that recalculates often enough to track temporal dynamics and adjust for the traffic demand of each item, thus overcoming the main difficulties of the method.

A recent interesting work in this line is Raven [12], which learns the distribution of content objects next-request arrival time from timestamped data and computes the probabilities of each one being the last one to arrive, making it an eviction candidate. Raven explicitly accounts for the stochastic, time-varying, and non-stationary nature of object arrival processes and exhibited good performance even on real-world traces. Nevertheless, Raven does not address jointly optimizing bandwidth consumption while simultaneously ensuring a sufficiently good user experience.

Finally, there is work on hierarchical caching. For instance, [16] proposes an adaptive scheme for multi-tier CDN structure and jointly addresses routing and allocation problems while incorporating the concept of traffic classes to contemplate costly inter-ISP traffic. The approach also uses Lagrange analysis. Nevertheless, it makes some simplifications that may not hold in reality, such as assuming full knowledge of the items' popularities, imposing fixed-size items, and a fixed transfer bandwidth from the devices for each request, which does not reflect how TCP works and changes the underlying optimization problems significantly.

Similarly, as shown in Table 1, [29] extends upon the previous work by allowing heterogeneous item sizes, while [9, 13, 14] does it by introducing considerations on parameters related to quality of experience. Also, [13, 14, 19] provide approximation methods capable of handling tens of thousands of devices (*10K scalability*). All these works assume known popularities, which is an unrealistic assumption. On the other hand, works addressing popularity prediction fail to consider the networking aspects of the problem [12, 24, 34, 40]. *Tero* extends upon these works by not only addressing all the previously mentioned elements together (e.g., routing, allocation, bandwidth considerations, sufficient scalability) but also reducing communication overhead with the load estimation module and solving the allocation fast enough.

6 CONCLUSIONS

We presented *Tero*, a system that performs fast cache routing and allocation decisions for highly distributed content delivery networks. It achieves high throughput from the CDN's lower tier, successfully offloading costly traffic from the regional CDN server while at the same time providing sufficient bandwidth to each request to provide a satisfactory user experience. *Tero* leverages content popularity predictions to drive the allocation decision process and performs these operations in seconds, starkly contrasting traditional approaches to the problem.

These advantages stem from crucial design choices, such as simple short-term predictions that leverage recent past behavior and can adapt quickly to changes, a novel optimal-properties-inspired heuristic allocation scheme, and an efficient mechanism to estimate the devices' current load and route requests accordingly. Evaluations on synthetic and real-world CDN traces showed that *Tero* can adapt to different workloads and consistently delivers good performance in terms of traffic throughput with respect to upper bounds. Finally, as future work, we plan to explore enhancements of the components by introducing machine learning techniques.

REFERENCES

[1] Umit Akinc and Basheer M Khumawala. 1977. An efficient branch and bound algorithm for the capacitated warehouse location problem. *Management Science* 23, 6 (1977), 585–594.

[2] Xuan Bao, Yin Lin, Uichin Lee, Ivica Rimac, and Romit Roy Choudhury. 2013. DataSpotting: Exploiting naturally clustered mobile devices to offload cellular traffic. In *2013 Proceedings IEEE INFOCOM*. 420–424. https://doi.org/10.1109/INFCOM.2013.6566807

[3] John E Beasley. 1988. An algorithm for solving large capacitated warehouse location problems. *European Journal of Operational Research* 33, 3 (1988), 314–325.

[4] L. A. Belady. 1966. A Study of Replacement Algorithms for a Virtual-Storage Computer. *IBM Syst. J.* 5, 2 (jun 1966), 78–101. https://doi.org/10.1147/sj.52.0078

[5] Daniel S. Berger. 2018. Towards Lightweight and Robust Machine Learning for CDN Caching. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks* (Redmond, WA, USA) *(HotNets '18)*. Association for Computing Machinery, New York, NY, USA, 134–140. https://doi.org/10.1145/3286062.3286082

[6] Livia Elena Chatzieleftheriou, Merkouris Karaliopoulos, and Iordanis Koutsopoulos. 2017. Caching-aware recommendations: Nudging user preferences towards better caching performance. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. 1–9. https://doi.org/10.1109/INFOCOM.2017.8057031

[7] Fabián A. Chudak and David P. Williamson. 1999. Improved Approximation Algorithms for Capacitated Facility Location Problems. In *Integer Programming and Combinatorial Optimization*, Gérard Cornuéjols, Rainer E. Burkard, and Gerhard J. Woeginger (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 99–113.

[8] Gérard Cornuéjols, Ranjani Sridharan, and Jean-Michel Thizy. 1991. A comparison of heuristics and relaxations for the capacitated plant location problem. *European journal of operational research* 50, 3 (1991), 280–297.

[9] Mostafa Dehghan, Bo Jiang, Anand Seetharam, Ting He, Theodoros Salonidis, Jim Kurose, Don Towsley, and Ramesh Sitaraman. 2017. On the Complexity of Optimal Request Routing and Content Caching in Heterogeneous Cache Networks. *IEEE/ACM Trans. Netw.* 25, 3 (jun 2017), 1635–1648. https://doi.org/10.1109/TNET.2016.2636843

[10] A. Geoffrion and R. Me Bride. 1978. Lagrangean Relaxation Applied to Capacitated Facility Location Problems. *A I I E Transactions* 10, 1 (1978), 40–47. https://doi.org/10.1080/05695557808975181

[11] Dongsu Han, David Andersen, Michael Kaminsky, Dina Papagiannaki, and Srinivasan Seshan. 2011. Hulu in the neighborhood. In *2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011)*. 1–10. https://doi.org/10.1109/COMSNETS.2011.5716501

[12] Xinyue Hu, Eman Ramadan, Wei Ye, Feng Tian, and Zhi Li Zhang. 2022. Raven: Belady-Guided, Predictive (Deep) Learning for In-Memory and Content Caching. In *CoNEXT 2022 - Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*. Association for Computing Machinery, Inc, 72–90. https://doi.org/10.1145/3555050.3569134

[13] Lemei Huang, Sheng Cheng, Yu Guan, Xinggong Zhang, and Zongming Guo. 2020. Consistent User-Traffic Allocation and Load Balancing in Mobile Edge Caching. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 592–597. https://doi.org/10.1109/INFOCOMWKSHPS50562.2020.9162921

[14] Stratis Ioannidis and Edmund Yeh. 2018. Jointly Optimal Routing and Caching for Arbitrary Network Topologies. *IEEE Journal on Selected Areas in Communications* 36, 6 (2018), 1258–1275. https://doi.org/10.1109/JSAC.2018.2844981

[15] Akanksha Jain and Calvin Lin. 2018. Rethinking Belady's Algorithm to Accommodate Prefetching. In *Proceedings of the 45th Annual International Symposium on Computer Architecture* (Los Angeles, California) *(ISCA '18)*. IEEE Press, 110–123. https://doi.org/10.1109/ISCA.2018.00020

[16] Wenjie Jiang, Stratis Ioannidis, Laurent Massoulié, and Fabio Picconi. 2012. Orchestrating massively distributed CDNs. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. 133–144.

[17] Mathias Lecuyer, Joshua Lockerman, Lamont Nelson, Siddhartha Sen, Amit Sharma, and Aleksandrs Slivkins. 2017. Harvesting Randomness to Optimize Distributed Systems. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks* (Palo Alto, CA, USA) *(HotNets-XVI)*. Association for Computing Machinery, New York, NY, USA, 178–184. https://doi.org/10.1145/3152434.3152435

[18] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. 1999. On the Existence of a Spectrum of Policies That Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies. *SIGMETRICS Perform. Eval. Rev.* 27, 1 (may 1999), 134–143. https://doi.org/10.1145/301464.301487

[19] Boxi Liu, Konstantinos Poularakis, Leandros Tassiulas, and Tao Jiang. 2019. Joint Caching and Routing in Congestible Networks of Arbitrary Topology. *IEEE Internet of Things Journal* 6, 6 (2019), 10105–10118. https://doi.org/10.1109/JIOT.2019.2935742

[20] Bruce M. Maggs and Ramesh K. Sitaraman. 2015. Algorithmic Nuggets in Content Delivery. *SIGCOMM Comput. Commun. Rev.* 45, 3 (jul 2015), 52–66. https://doi.org/10.1145/2805789.2805800

[21] Dhruv Matani, Ketan Shah, and Anirban Mitra. 2021. An O (1) algorithm for implementing the LFU cache eviction scheme. *arXiv preprint arXiv:2110.11602* (2021).

[22] Sanjay Melkote and Mark S. Daskin. 2001. Capacitated facility location/network design problems. *European Journal of Operational Research* 129, 3 (2001), 481–495. https://doi.org/10.1016/S0377-2217(99)00464-6

[23] Pitu B Mirchandani and Richard L Francis. 1990. *Discrete location theory.*

[24] Arvind Narayanan, Saurabh Verma, Eman Ramadan, Pariya Babaie, and Zhi-Li Zhang. 2018. DeepCache: A Deep Learning Based Framework For Content Caching. In *Proceedings of the 2018 Workshop on Network Meets AI & ML* (Budapest, Hungary) *(NetAI'18)*. ACM, 48–53. https://doi.org/10.1145/3229543.3229555

[25] Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. 1993. The LRU-K Page Replacement Algorithm for Database Disk Buffering. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* (Washington, D.C., USA) *(SIGMOD '93)*. Association for Computing Machinery, New York, NY, USA, 297–306. https://doi.org/10.1145/170035.170081

[26] Georgios Paschos, George Iosifidis, Giuseppe Caire, et al. 2020. Cache Optimization Models and Algorithms. *Foundations and Trends® in Communications and Information Theory* 16, 3–4 (2020), 156–345.

[27] Georgios S. Paschos, Apostolos Destounis, Luigi Vigneri, and George Iosifidis. 2019. Learning to Cache With No Regrets. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications* (Paris, France). IEEE Press, 235–243. https://doi.org/10.1109/INFOCOM.2019.8737446

[28] Konstantinos Poularakis, George Iosifidis, and Leandros Tassiulas. 2014. Approximation Algorithms for Mobile Data Caching in Small Cell Networks. *IEEE Transactions on Communications* 62, 10 (2014), 3665–3677. https://doi.org/10.1109/TCOMM.2014.2351796

[29] Konstantinos Poularakis, Jaime Llorca, Antonia M. Tulino, Ian Taylor, and Leandros Tassiulas. 2020. Service Placement and Request Routing in MEC Networks With Storage, Computation, and Communication Constraints. *IEEE/ACM Transactions on Networking* 28, 3 (2020), 1047–1060. https://doi.org/10.1109/TNET.2020.2980175

[30] Kaushik Rajan and Govindarajan Ramaswamy. 2007. Emulating Optimal Replacement with a Shepherd Cache. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. 445–454. https://doi.org/10.1109/MICRO.2007.25

[31] Alireza Sadeghi, Fatemeh Sheikholeslami, and Georgios B. Giannakis. 2018. Optimal and Scalable Caching for 5G Using Reinforcement Learning of Space-Time Popularities. *IEEE Journal of Selected Topics in Signal Processing* 12, 1 (2018), 180–190. https://doi.org/10.1109/JSTSP.2017.2787979

[32] Sandvine. 2023. 2023 Global Internet Phenomena Report. online. accesed on 18 Jun 2023.

[33] Qiquan Shi, Jiaming Yin, Jiajun Cai, Andrzej Cichocki, Tatsuya Yokota, Lei Chen, Mingxuan Yuan, and Jia Zeng. 2020. Block Hankel tensor ARIMA for multiple short time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 5758–5766.

[34] Zhenyu Song, Daniel S Berger, Kai Li, Anees Shaikh, Wyatt Lloyd, Soudeh Ghorbani, Changhoon Kim, Aditya Akella, Arvind Krishnamurthy, Emmett Witchel, et al. 2020. Learning relaxed belady for content distribution network caching. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 529–544.

[35] Ramaswami Sridharan. 1995. The capacitated plant location problem. *European Journal of Operational Research* 87, 2 (1995), 203–213.

[36] Volker Stocker, Georgios Smaragdakis, William Lehr, and Steven Bauer. 2017. The Growing Complexity of Content Delivery Networks: Challenges and Implications for the Internet Ecosystem. *Telecommunications Policy Journal* 41, 10 (November 2017), 1003–1016.

[37] Linpeng Tang, Qi Huang, Amit Puntambekar, Ymir Vigfusson, Wyatt Lloyd, and Kai Li. 2017. Popularity prediction of facebook videos for higher quality streaming. In *USENIX Annual Technical Conference.*

[38] Alexandru Tatar, Marcelo Dias De Amorim, Serge Fdida, and Panayotis Antoniadis. 2014. A survey on predicting the popularity of web content. *Journal of Internet Services and Applications* 5, 1 (2014), 1–20.

[39] A. Vakali and G. Pallis. 2003. Content delivery networks: status and trends. *IEEE Internet Computing* 7, 6 (2003), 68–74. https://doi.org/10.1109/MIC.2003.1250586

[40] Gang Yan and Jian Li. 2020. RL-Bélády: A Unified Learning Framework for Content Caching. In *Proceedings of the 28th ACM International Conference on Multimedia* (Seattle, WA, USA) *(MM '20)*. Association for Computing Machinery, New York, NY, USA, 1009–1017. https://doi.org/10.1145/3394171.3413524

[41] Gang Yan, Jian Li, and Don Towsley. 2021. Learning from Optimal Caching for Content Delivery *(CoNEXT '21)*. Association for Computing Machinery, New York, NY, USA, 344–358. https://doi.org/10.1145/3485983.3494855

[42] Behrouz Zolfaghari, Gautam Srivastava, Swapnoneel Roy, Hamid R. Nemati, Fatemeh Afghah, Takeshi Koshiba, Abolfazl Razi, Khodakhast Bibak, Pinaki Mitra, and Brijesh Kumar Rai. 2020. Content Delivery Networks: State of the Art, Trends, and Future Roadmap. *ACM Comput. Surv.* 53, 2, Article 34 (apr 2020), 34 pages. https://doi.org/10.1145/3380613

A APPENDIX

A.1 Model's notation

Table 2 summarizes the notation used in the problem.

A.2 Indirect detection of excessive concurrency

To perform detection of excessive concurrency, the CEM uses a threshold on the makespan estimation. All devices with makespan above the threshold are declared overloaded and excluded from the routing process to avoid generating excess traffic.

The threshold $\frac{\hat{s}_d}{2\delta}$ comes from assuming random arrivals at unknown times. Proof sketch: When one checks the makespan of a fair-queue sharing device with average item size $\hat{s}$ and no information whatsoever of when each request started to be transmitted, one assumes an i.i.d. uniformly random distribution over the remaining fraction to be transmitted for each ongoing request: z_k for request k for r concurrent requests. Value r is unknown. Then, $z_k \sim U[0,1] \forall k$. Let $z = \hat{s}\sum_{k=1}^{r} z_k$ be the total number of bytes that remain to be transmitted.

Note that $z/\hat{s}$ follows an Irwin distribution of parameter r. Therefore, on expectation:

$$\mathbb{E}[z] = \mathbb{E}[\sum_k \hat{s} z_k] = \hat{s}\sum_k \mathbb{E}[z_k] = \frac{\hat{s}r}{2}$$

Table 2: Notation of problem variables.

$i \in [M]$	content object or item index.
s_i	item size.
p_i	item popularity: expected number of requests per time unit.
$d \in [D]$	caching device index.
$d = 0$	the server.
C_d	device's storage capacity
B_d	device's upload bandwidth
σ	Request sequence up to time T. Each request $\sigma_n \in [M]$.
$\lambda = \frac{N}{T}$	request process intensity.
H	throughput towards the users.
H_c	throughput from the cache devices.
H_s	throughput from the server.
$X = \{x_{id}\}$	Allocation matrix.
$x_{id} \in \{0,1\}$	$x_{id} = 1$ if item i is allocated on device d ($i \in d$).
$A = \{\alpha_{id}\}$	Routing matrix.
$\alpha_{id} \in [0,1]$	fraction of item i's traffic demand served from d.
N_{id}	Number of requests for item i served from d.
δ	Minimum bandwidth per request.
r_d	number of concurrent requests at d
R_d	maximum allowed concurrent requests at d.

Now we want to make obtain the most likely value $\hat{r}$ for an estimation of r, i.e., $\hat{r} = \arg\max \text{likelihood}(r \mid z)$, which for a single data point results in $\hat{r} = \arg\max z|r$. As Irwin(r) is a unimodal distribution, symmetric around its mean $r/2$, then $\hat{r} \approx 2z/\hat{s}$. Taking m and B as the device's makespan and bandwidth respectively, we can write $\hat{r} \approx 2mB/\hat{s}$. So the condition $r \leq R = B/\delta$ is approximately satisfied when $m \leq \frac{\hat{s}}{2\delta}$, which is the threshold CEM uses.

A.3 Insights on system model

The model in Section 2.2 is a variant of the classical Capacitated Facility Location Problem (CFLP, [1, 3, 22]). On its basic version, the problem consists of choosing facility locations to minimize transportation costs towards customers, subject to constraints requiring demands to be serviced by the established facilities. In our model, items are mapped to CFLP customers, devices to facilities, and costs to sizes.

Our model differs from classical CFLP in the many aspects. First, our version is doubly capacitated (in both bandwidth and storage). Second, it introduces and admission constraint (QoE) which is basically online in nature and constitutes an additional, novel obstacle.

The CFLP problem is known to be NP-hard, and in general also NP-hard to approximate, although under mild conditions there exist approximation heuristics in polynomial time [7].

Already, joint routing and allocation for CDN was presented as a CFLP in [26], in which it was also shown that under this context the problem is NP-hard to approximate. Nevertheless, that formulation does not have quality-of-experience-based constraints.

As far as we know, we are the first ones to address a CFLP with this particular set of constraints. As a consequence, we deem the original problem formulation as a hard problem which in order to be solved fast would require a novel approach that we address in the next section.

A.4 SDM optimal allocation

The model for the SDM can be derived from *Tero*'s model by replacing $D = 1$, the corresponding bandwidth B and capacity C and realizing that there is no routing problem. Even more, given that the SDM is intended as an upper bound on *Tero*, we also remove the user satisfaction constraint to achieve even higher traffic. Finally, we also simplify the notation and use $y_i = x_{i1}$ to get: the resulting SDM model:

$$\max_y \sum_{i=1}^{M} s_i p_i y_i \tag{19}$$

$$\text{subject to: } \sum_{i=1}^{M} y_i s_i \leq C \tag{20}$$

$$y_i \in \{0,1\} \forall i \in [M] \tag{21}$$

Which is a knapsack problem. We relax the integrality condition and using the Lagrangian formulation:

$$\mathcal{L} = -\sum_{i=1}^{M} s_i p_i y_i + \lambda_0(\sum_{i=1}^{M} y_i s_i - C) + \sum_{i=1}^{M} \lambda_i y_i (y_i - 1)$$

where the factors λ_i are non-negative Lagrangian multipliers, λ_0 corresponding to the storage constraint.

On one side we know that

$$\lambda_i \geq 0 \, \forall i \in [0, M]$$

and that the stationary condition becomes:

$$\frac{\partial \mathcal{L}}{\partial y_i} = -s_i p_i + \lambda_0 s_i + \lambda_i (2y_i - 1) = 0$$

So,

$$s_i p_i = \lambda_0 s_i + \lambda_i (2y_i - 1)$$

Assume the SDM has remaining space. Then $\lambda_0 = 0$. As the left side of this expression is non-negative and by equation A.4 we know that $\lambda_i > 0$ and $2y_i - 1 > 0$. This means that $y_i(y_i - 1) = 0$, ie, $y_i \in \{0, 1\}$. Now then $2y_i - 1$ is positive only when $y_i = 1$. We have to conclude that $y_i = 1$ for all i, which is an impossibility as there is not enough space in the device to hold all objects. The exact same analysis holds if we just considered $\lambda_0 = 0$. So this case is infeasible and we must conclude that $\lambda_0 > 0$ and the optimal solution the device is filled up to its storage capacity.

Now consider any object i such that $p_i > \lambda_0$. Then to satisfy the stationary condition we need $y_i = 1$. Analogously, if $p_i < \lambda_0$, $y_i = 0$. So if the SDM is allocated by item's popularity ranking until no other item fits in the storage space, we can approximate the optimal solution.