RED/LED: An Asymptotically Optimal and Scalable Online Algorithm for Service Caching at the Edge

Tao Zhao, I-Hong Hou, Shiqiang Wang, and Kevin Chan

***Abstract*—Edge servers, which are small servers located close to mobile users, have the potential to greatly reduce delay and backhaul traffic of mobile Internet applications by moving cloud services to the edge of the network. Due to limited capacity of edge servers and dynamic request arrival, proper service caching at the edge is essential to guarantee good performance. This paper proposes a tractable online algorithm called *retrospective download with least-requested deletion* (RED/LED) that caches services dynamically without any assumptions on the arrival patterns of mobile applications. We evaluate the competitive ratio of our policy, which quantifies the worst-case performance in comparison to an optimal offline policy. We prove that the competitive ratio of our policy is linear with the capacity of the edge server. We also show that no deterministic online policy can achieve a competitive ratio that is asymptotically better than ours. Moreover, we prove that our policy is scalable, in the sense that it only needs doubled capacity to achieve a constant competitive ratio. The utility of our online policy is further evaluated on real-world traces. These trace-based simulations demonstrate that our policy has better, or similar, performance compared to many intelligent offline policies.**

***Index Terms*—Competitive ratio, edge computing, online algorithms, optimal scheduling, service caching.**

I. INTRODUCTION

MANY emerging mobile applications rely on cloud computing technology to greatly expand the capability of resource-constrained mobile devices. In a typical scenario, a mobile device sends a request, such as a picture containing text in a foreign language, to a remote cloud, which is often hosted by a remote data center. The remote cloud then generates a response, such as translations of the text, using its massive computational power and storage. However, the long distance between mobile devices and remote clouds can result in significant delay and severe burden on the backhaul connection, which can limit the development of real-time and data-intensive applications. The concept of edge computing, also known as cloudlets, fog computing, etc., has been proposed to address this issue [2]–[4]. In edge computing, small edge servers are deployed close to mobile users, such as at the locations of cellular base stations or WiFi access points. These edge servers can host a small number of popular services, and provide timely response to local user requests directly without communicating with remote clouds. Edge servers are not necessarily real "servers". They can be, for example, home WiFi routers in smart home environment, as suggested in [5].

Edge servers have limited computational power and storage compared to remote clouds. Thus, they can only "cache" a small number of services, out of all the available services hosted by the remote cloud [6], [7]. In this paper, we say that a service is *cached* at the edge server when the entire set of code and data required for service execution has been downloaded from the remote cloud to the edge server. The edge server can fully execute a cached service on its own, without interacting with the remote cloud. Since the arrivals of service requests from mobile devices can be highly dynamic, proper service caching at the edge is essential to guarantee good performance, and it is challenging to find the optimal caching policy that determines which services to cache at the edge server.

A. Motivation

While there have been studies on optimal service caching at the edge, most of them assume either that the edge servers have reasonably good predictions about future requests [8], [9], or that the arrivals of requests follow a pre-specified stochastic process [10]–[14]. These studies then use predictions or stochastic models to determine the services that the edge servers should cache to achieve the optimal average performance. However, newly emerging services can be challenging to predict or model in advance. Besides, as suggested by a real-world trace of requests at the cloud [15], the request arrival patterns can change frequently over time, which are difficult to timely predict or model as a known stochastic process. In addition, predictions are generally imperfect. Algorithms that rely on predictions or stochastic models can result in poor performance for these systems. Further, many mission-critical systems require "worst-case" performance guarantees, instead of "average" performance guarantees.

Tao Zhao and I-Hong Hou are with Department of ECE, Texas A&M University, College Station, Texas 77843-3128, USA. Email: {alick, ihou}@tamu.edu

Shiqiang Wang is with IBM T.J. Watson Research Center, Yorktown Heights, NY, USA. Email: wangshiq@us.ibm.com

Kevin Chan is with US Army Research Laboratory, Adelphi, MD, USA. Email: kevin.s.chan.civ@mail.mil

This material is based upon work supported in part by, NSF under contract number CNS-1719384, Office of Naval Rsearch under Contract N00014-18-1-2048, the U.S. Army Research Laboratory and the U.S. Army Research Office under contract/grant number W911NF-15-1-0279, the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

Part of this work has been presented at ACM MobiHoc 2016 [1].

In this paper, we study online algorithms that determine which services to cache at the edge dynamically without making any assumptions on the arrival patterns of requests. We focus on deterministic policies since they have predictable system behavior and are generally more robust in practice. We consider an edge server that can cache only a limited number of services. When a request arrives but its service is not cached at the edge server, the edge server has two options: It can either forward the request to the remote cloud for processing, or it can download and cache the service so that it can directly serve the requests for this service in the future. Both options have a cost, which can reflect the delay and the network bandwidth usage. We assume the cost of downloading a service is larger than the cost of forwarding a request. This is motivated by the fact that forwarding a request to a remote cloud for processing has a response time that is usually within a second, as shown by the experimental results presented in [16]. Downloading a service, which entails both downloading necessary files and setting up a virtual machine or container, takes at least several seconds according to recent experimental studies [17]–[19].

We aim to design online algorithms that result in a small total cost, including the cost of forwarding requests and downloading services, for any sequence of request arrivals. As online algorithms have no knowledge about future arrivals, we evaluate the performance of an online policy by its *competitive ratio*, defined as the largest possible ratio between the cost of the online policy and the minimum (optimal) offline cost, under any sequence of arrivals. While this problem may seem to bear some similarities with the classic data caching problem, we note that the option of forwarding a request, and the significant difference between the costs of forwarding a request and downloading a service, make the problem of service caching fundamentally different from the problem of data caching. In fact, as we will demonstrate in Section IX, the Belady's algorithm, a well-known optimal offline policy for data caching, performs poorly for service caching.

B. Main Contribution

We first focus on a homogeneous system where all services require the same amount of computational power and storage, and they have the same forward cost as well as the same download cost.[1] Using an observation of the optimal offline policy, we propose an online policy, *retrospective download with least-requested deletion* (RED/LED), for service caching at the edge, which is easy to implement in practice. We prove that the competitive ratio of our RED/LED policy only grows linearly with the capacity of the edge server. We further prove that no deterministic online policy can achieve a competitive ratio that is sublinear with the capacity of the edge server. Therefore, our RED/LED policy indeed achieves the optimal asymptotic performance. Moreover, we prove that our policy is scalable, in the sense that if the capacity of the edge server is doubled for RED/LED, it can achieve a constant competitive ratio with regard to the optimal offline batch-download policy.

We then address several practical issues of RED/LED. We demonstrate that it can be implemented as an algorithm with linear time complexity. We also propose an extension of RED/LED for heterogeneous systems where different services may be associated with different costs and require different amounts of edge-server resources.

We evaluate the performance of our RED/LED policy using real-world traces of service request arrivals. We compare our policy against a randomized online policy, the optimal offline batch-download policy, and three other offline policies that correspond to solutions based on dynamic programming, stochastic optimization, and data caching respectively. We note that the time complexity of the optimal offline algorithm is very high, and all the offline policies have complete knowledge of all future arrivals. Each of the offline policies achieves the optimal performance under some specific scenarios. Simulation results show that our policy achieves better, or at least similar, performance compared to all these policies in both homogeneous and heterogeneous systems.

The rest of the paper is organized as follows. Section II reviews related studies. Section III formally describes the service caching problem. Section IV introduces our online policy based on a property of the optimal offline policy. Section V derives the competitive ratio of our policy. Section VI proves that no deterministic online policy can be asymptotically better than ours. Section VII shows the scalability of our policy. Section VIII addresses practical issues including heterogeneous systems and low complexity implementation. Section IX compares our policy against several others through trace-based simulations. Finally, Section X concludes the paper.

II. Related Work

The dramatic increase in network traffic, particularly due to the proliferation of mobile devices and Internet of Things (IoT), has made it critical to move some computing and storage jobs from remote data centers to the edge of the network. There have been a number of architecture proposals of edge computing in the literature [2]–[4], [7], [20]–[22]. The utility of edge computing has been demonstrated by various prototypes [5], [23], [24]. Comprehensive surveys on this new paradigm have been published recently [25], [26].

To address the challenge of managing limited resources of edge servers, some studies rely on an accurate prediction of future requests. Tadrous et al. [8] have considered systems where one can predict the popularity of services, so that edge servers can proactively download and replicate popular services during off-peak hours. Llorca et al. [9] have studied the content placement problem and proposed an optimal offline policy. Wang et al. [21] have proposed an online algorithm with polynomial time complexity to minimize the long-term average cost. Yang et al. [27] have studied the joint optimization of service placement and load dispatching. Unlike these existing studies, in this paper we investigate online policies which assume no knowledge about future requests.

Besides, there are many studies that employ stochastic optimization for service caching at the edge. Amble et al. [10] have considered a system where request arrivals follow an independent and identically distributed (i.i.d.) stochastic process with unknown distribution, and proposed a stochastic

[1]The download cost is different from the forward cost.

control policy that maximizes the capacity region of the system. Borst et al. [12] have proposed an algorithm for a static system where the popularity of services does not change over time. On the other hand, Wang et al. [13] and Urgaonkar et al. [14] have considered dynamic systems where request arrivals are modeled as a Markov process, and proposed solutions that achieve the optimal long-term average performance. Qiu et al. [11] have employed Lyapunov optimization to maximize the performance of edge servers. However, these solutions based on stochastic optimization cannot provide "worst-case" performance guarantees with regard to an optimal offline policy. Our work, on the other hand, addresses such issues.

The problem of service caching at the edge bears some similarities with the classic data caching problem in computer architecture. In the data caching problem, requests for data arrive sequentially, and a "miss" occurs when the data is not stored in the cache. The requested data is then downloaded, and a policy needs to decide which data to delete to minimize the total number of misses. When one has complete knowledge of all future requests, the Belady's algorithm [28] achieves the optimal performance. As for online policies without any knowledge of future requests, Sleator and Tarjan [29] have established a least recently used (LRU) policy that is optimal among all deterministic policies. They have also showed that LRU has a constant competitive ratio if its cache capacity is a constant factor of that of the optimal offline policy. Achlioptas et al. [30] have studied the competitive ratios of randomized policies. Bansal et al. have investigated the weighted case where the download costs vary for different data [31]. Despite the similarities, there are notable differences between the data caching problem and the service caching problem. In the data caching problem, one cache miss implies one download with some cost, and forwarding is disallowed. However, in the service caching problem, forwarding is permitted and incurs a smaller cost than downloading, and a policy needs to decide whether to download a service in addition to which service to delete.

III. System Model

We consider an edge-cloud system as illustrated in Fig. 1. An edge server and a back-end cloud are connected through a backhaul connection. The edge server and the back-end cloud jointly host a set $\mathbb{S}$ of services, numbered as $S_1, S_2, \ldots$. The services in the system include face detection, video streaming, translation, smart home services and so on [5], [17]. The back-end cloud has massive capacity, and can host all services. On the other hand, the edge server has limited capacity and can only cache K services. Without loss of generality, we assume that, when the system starts, the edge server caches services $S_1, S_2, \ldots, S_K$.

Requests for services arrive at the edge server sequentially. Requests from different mobile devices can arrive at and be processed by the edge server either on a first-come-first-served basis, or based on some other scheduling mechanism using queues/buffers. We use $r_n \in \mathbb{S}$ to denote the service requested by the n-th request. If r_n is cached by the edge server when the request arrives, then the edge server can serve the request

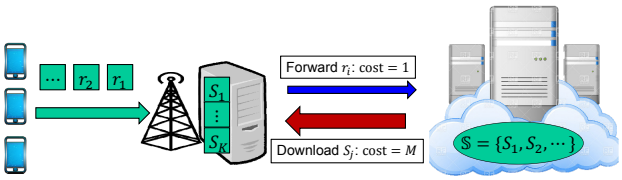


Fig. 1. An illustration of the edge-cloud system. The back-end cloud can host all services $\mathbb{S}$, while the edge server can only cache a subset of size K. Requests can incur different costs depending on which services are cached at the edge.

immediately without causing any significant cost. On the other hand, if r_n is not cached by the edge server, then the edge server has two choices: First, it can forward this request to the back-end cloud for processing. This will cause some delay as well as some traffic on the backhaul connection. We say that forwarding a request to the back-end cloud incurs a cost of one unit. Second, the edge server can instead download and replicate the whole service r_n and serve the request. Downloading a service can cause much higher delay and more traffic. Therefore, we say that each service download incurs a cost of M units, with $M \geq 1$. In practice, we can choose M as the ratio of the average delay of downloading to the average delay of forwarding. On the other hand, since the edge server caches the service r_n after the download, it can then serve subsequent requests for r_n without incurring any costs. The edge server can only cache K services. Therefore, when it downloads a service, it also needs to delete a service from its storage.

We aim to minimize the total cost of the system by intelligently reconfiguring the set of services cached by the edge server. Intuitively, if we know that a service will have a lot of requests in the near future, we should download this service so that all these requests only incur M units of cost. Otherwise, we should simply forward all these requests to the back-end cloud without downloading the service, and pay one unit of cost for each request. In practice, however, we may not have accurate prediction for future requests. In such cases, we need to rely on online policies that assume no information about future requests.

Let OPT be the optimal offline policy that minimizes the total cost of the system, which has full information about all future request arrivals. Let η be an online policy that makes its decision solely based on past events. For a given sequence of request arrivals, $r_1, r_2, \ldots$, let C_{OPT} be the total cost of OPT, and C_η be the total cost of η. Note that the total costs, C_{OPT} and C_η, are functions of the sequence $r_1, r_2, \ldots$, but we omit the sequence to simplify the notation. There may be multiple offline policies that achieve the minimum cost. In this case, we let OPT be one that makes the most downloads among them. We evaluate the performance of an online policy η by its *competitive ratio*, which is the largest possible value of C_η / C_{OPT}, over all possible sequences of request arrivals.

Definition 1 (Competitive Ratio): An online policy η is said to be β-competitive, or have a competitive ratio of β, if $C_\eta \leq \beta C_{\text{OPT}}$, for every possible sequence of request arrivals.

An online policy with a low competitive ratio has similar

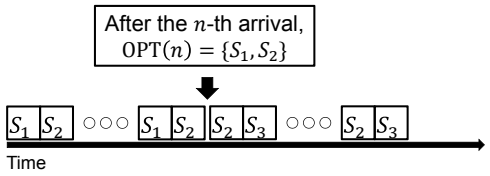


Fig. 2. An example illustrating operations of OPT and RED/LED. The first n arrivals are $S_1, S_2, \ldots, S_1, S_2$, and the next $4M$ arrivals are $S_2, S_3, \ldots, S_2, S_3$.

performance with the optimal offline policy. Therefore, we aim to develop an online policy with a low competitive ratio, as well as a lower bound of competitive ratios for all online policies.

For brevity, we say that a policy caches a service if, under the said policy, the edge server caches the service. To facilitate the analysis, let $\zeta(n) \subset \mathbb{S}$ denote the subset of services cached by a policy ζ, online or offline, after the n-th arrival, for a given sequence of request arrivals, $r_1, r_2, \ldots$. Again, note that $\zeta(n)$ is a function of the sequence of arrivals. For a service S_i, we use $x_i(n) := \mathbb{1}\{r_n = S_i\}$ to indicate whether or not the n-th request is for this service. For brevity, let $[n, m]$ denote the inclusive time interval between the n-th and the m-th arrival.[2] Therefore, $\sum_{l=n}^{m} x_i(l)$ is the number of requests for S_i during $[n, m]$.

Before proceeding to the next section, we note that the system model in this section is a homogeneous one: All services have the same cost of forwarding requests, and the same cost of downloading. Moreover, all services require the same amount of edge-server resources. While our analytical results mainly focus on the homogeneous system, we will show that our policy can be easily extended to heterogeneous systems in Section VIII, and it works very well in evaluation in Section IX.

IV. The RED/LED Online Policy

In this section, we first establish a basic property of OPT, and then use it to develop our online policy RED/LED.

A. A Basic Property of OPT

Theorem 1: Suppose we are given a sequence $r_1, r_2, \ldots$, and $\text{OPT}(n)$, which is the subset of services cached by OPT after the n-th arrival. If there exists an integer $m > n$, a service $S_i \notin \text{OPT}(n)$, and another service $S_j \in \text{OPT}(n)$ such that:

$$\sum_{l=n+1}^{m} x_i(l) \geq \sum_{l=n+1}^{m} x_j(l) + 2M, \tag{1}$$

then OPT downloads at least one service during $[n+1, m]$.

Before proving Theorem 1, we first use Fig. 2 for illustration. Suppose the first n arrivals are $S_1, S_2, \ldots, S_1, S_2$, and the edge server can cache two services. After the n-th arrival, OPT caches S_1 and S_2. Between the $(n+1)$-th arrival and the $(n+4M)$-th arrivals, we have $2M$ requests for S_3 and no requests for S_1, that is, $\sum_{l=n+1}^{n+4M} x_1(l) = 0$, and $\sum_{l=n+1}^{n+4M} x_3(l) = 2M$. Theorem 1 then states that OPT downloads at least one service during $[n+1, n+4M]$. Note that Theorem 1 does not specify which service to download, and which service to delete from the edge server.

Proof of Theorem 1: We now prove Theorem 1 by contradiction. Suppose OPT does not download any service, and therefore does not delete any service, during $[n+1, m]$. That is, $\text{OPT}(l) = \text{OPT}(n)$, for all $n+1 \leq l \leq m$.

We construct a different policy ξ as follows: ξ caches the same subset of services as OPT before the n-th arrival and after the $(m+1)$-th arrival, that is, $\xi(l) = \text{OPT}(l)$, for all $l \leq n$, and all $l \geq m+1$. After the n-th arrival, ξ downloads S_i and deletes S_j so that $\xi(n+1) = \text{OPT}(n) \setminus \{S_j\} \cup \{S_i\}$. After the m-th arrival, ξ downloads S_j and deletes S_i, and then follows the same decisions that OPT makes so that $\xi(m+1) = \text{OPT}(m+1)$.

We now compare the costs of ξ and OPT. Since ξ and OPT are the same before the n-th arrival and after the m-th arrival, they incur the same amount of costs in these two durations. Between the n-th arrival and the m-th arrival, ξ downloads two services and OPT downloads none. Therefore, ξ needs to pay $2M$ units of cost. Meanwhile, ξ needs to pay one unit cost for each request for S_j, and there are $\sum_{l=n+1}^{m} x_j(l)$ of them, while OPT needs to pay one unit cost for each of the $\sum_{l=n+1}^{m} x_i(l)$ requests for S_i. The two policies incur the same amount of costs for all other requests. In summary, we have:

$$C_\xi = C_{\text{OPT}} + 2M + \sum_{l=n+1}^{m} x_j(l) - \sum_{l=n+1}^{m} x_i(l) \leq C_{\text{OPT}},$$

where the last inequality follows by (1). Therefore, ξ also minimizes the total cost. Moreover, ξ makes two more downloads than OPT. Recall that OPT is defined as the policy that makes the most downloads among all policies with minimum cost. The existence of ξ therefore contradicts with assumptions of OPT. ■

B. The RED/LED Online Policy

We now introduce our online policy. The policy needs to consist of two parts: deciding whether to download a service, and, when a download occurs, deciding which service to delete from the edge server. We propose the *retrospective download with least-requested deletion* (RED/LED) policy that uses a *retrospective download* (RED) policy for the first part, and a *least-requested deletion* (LED) policy for the second part. We use $\text{RL}(n)$ to denote the subset of services cached by RED/LED after the n-th request arrival.

RED is used to determine whether to download a service, and is formally defined as follows:

*Definition 2 (*RED*):* When a request $r_n = S_i \notin \text{RL}(n-1)$ arrives, RED downloads and replicates S_i at the edge server if there exists an integer τ and a service S_j such that for all $n - \tau \leq l \leq n-1$, $S_j \in \text{RL}(l)$ and $S_i \notin \text{RL}(l)$, and

$$\sum_{l=n-\tau}^{n} x_i(l) \geq \sum_{l=n-\tau}^{n} x_j(l) + 2M. \tag{2}$$

The intuition of RED comes from Theorem 1. Suppose $S_j \in \text{OPT}(n-\tau)$, $S_i \notin \text{OPT}(n-\tau)$, and (2) holds. Then

[2]Throughout this paper, $[n, m] = \{n, n+1, \ldots, m\}$. Time refers to indices of request arrivals, which are not necessarily physical time.

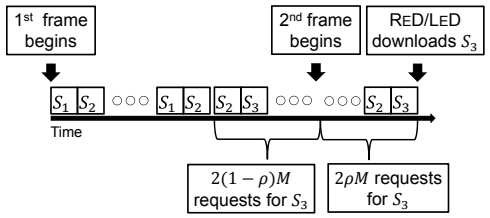


Fig. 3. An example for dividing the costs into frames.

Theorem 1 states that OPT downloads at least one service between the $(n-\tau)$-th arrival and the n-th arrival. RED then downloads S_i when, in retrospect, it finds OPT would have already downloaded a service.

When RED decides to download a service, we need to choose a service to delete from the edge server. We propose a *least-requested deletion* (LED) policy as follows:

*Definition 3 (*LED*):* Suppose the edge server decides to download a service at the n-th arrival. For each service S_i currently cached by the edge server, let τ_i be the smallest integer such that there are at least $2M$ requests for S_i in the past τ_i arrivals, that is, $\sum_{l=n-\tau_i}^{n} x_i(l) \geq 2M$. LED deletes the service with the largest τ_i.

A service with larger τ_i needs to go further back in time to find at least $2M$ requests. LED therefore deletes the service that is least requested starting from $n-\tau_i$.

RED/LED uses RED to decide whether to download a service, and LED to decide which service to delete. We use Fig. 2 to illustrate the operation of RED/LED. Suppose $\text{RL}(n) = \{S_1, S_2\}$, that is, RED/LED caches S_1 and S_2 after the n-th arrival. At the $(n+4M)$-th arrival, we find that $\sum_{l=n+1}^{n+4M} x_3(l) = 2M \geq \sum_{l=n+1}^{n+4M} x_1(l) + 2M$, and RED decides to download S_3 at the $(n+4M)$-th arrival. Meanwhile, during $[n+1, n+4M]$, there are $2M$ requests for S_2 and none for S_1. We then have $\tau_1 > \tau_2$, and LED decides to delete S_1 to accommodate S_3.

V. The Competitive Ratio of RED/LED

This section establishes the competitive ratio of RED/LED by proving the following theorem:

Theorem 2: RED/LED is $10K$-competitive, where K is the number of services that can be cached by the edge server.

A. Overview of the Proof

We will compare the performance of RED/LED and OPT. Given OPT and the arrival sequence, we can divide the arrival sequence into frames, $[t_1+1, t_2], [t_2+1, t_3], \ldots$, so that OPT downloads services only at the beginning of frames, i.e., after the t_1-th, t_2-th, $\ldots$, arrivals. Before the first download under OPT, there must be no download under RED/LED due to its retrospective nature. Therefore, RED/LED and OPT behave the same during $[1, t_1]$, and below we will focus on the performance in frames $[t_1+1, t_2], [t_2+1, t_3], \ldots$.

We will define the costs of OPT and RED/LED in each frame. We define the cost of OPT in a frame as the sum of the download cost at the beginning of the frame and all the costs of forwarding requests to the back-end cloud during the frame. On the other hand, it can be challenging to define the cost of RED/LED in each frame properly. Consider the example in Fig. 3. RED/LED downloads S_3, and incurs M units of download cost, shortly after the second frame begins. The decision to download S_3 actually involves many requests in the first frame. It is then unreasonable to count all the M units of download cost against the second frame. Instead, we separate the M units of download cost as follows: Suppose there are only $2\rho M$, where $\rho < 1$, requests for S_3 in the second frame when RED/LED downloads it, we say that the download is a *partial download* of fraction ρ, and only incurs ρM units of cost. At the same time, another partial download of fraction $(1-\rho)$ occurs at the end of the first frame, and incurs $(1-\rho)M$ units of cost. This separation does not change the total cost of RED/LED. To further simplify the notation in the next subsection, we say that, at the end of a frame, all services not cached by RED/LED have a partial download, possibly with fraction 0. The cost of RED/LED in a frame will then consist of all the download costs, including those from partial downloads, and all the costs of forwarding requests to the back-end cloud, in this frame.

Below, we will calculate the costs of OPT and RED/LED in each frame, and prove Theorem 2 by showing that RED/LED incurs at most $10K$ times as much cost as OPT in each frame.

B. Costs in a Frame

Without loss of generality, we calculate the costs in a frame $[t_g+1, t_{g+1}]$. We use $C_{\text{OPT}}(g)$ and $C_{\text{RL}}(g)$ to denote the costs of OPT and RED/LED in this frame, respectively.[3]

We first consider a service $S_i \in \text{OPT}(t_g+1)$. In the frame, let $E_i \in \mathbb{Z}$ be the number of times RED/LED downloads S_i, and $D_i \in \mathbb{Z}$ be the number of times RED/LED deletes S_i.[4] Note we have $D_i \geq E_i - 1$, as a service needs to be deleted first in order to be downloaded again. Let $f_{i,z}$ be the number of requests for S_i that RED/LED forwards to the back-end cloud within the z-th interval of S_i not being cached at the edge server, i.e., after the previous deletion and before the next download of S_i, in this frame. Fig. 4 illustrates an example of the downloads and deletions of S_i in a frame, as well as the definition of $f_{i,z}$. S_i then incurs at most

$$E_i M + \sum_z f_{i,z} \tag{3}$$

units of cost under RED/LED.

On the other hand, we have the following lemma for OPT:

Lemma 1: Suppose there exist integers $n < m$ and a service S_i such that for all $n \leq l \leq m$, $\text{OPT}(l) = \text{OPT}(n)$, $S_i \in \text{OPT}(l)$, and $S_i \notin \text{RL}(l)$, then the number of requests that OPT forwards to the back-end cloud during $[n, m]$ is at least $\sum_{l=n}^{m} x_i(l) - 4M$.

Proof: See Appendix A. ∎

By Lemma 1, if $f_{i,z} > 4M$ for some i, z, then during the time of these $f_{i,z}$ requests for S_i, OPT forwards at least $f_{i,z} -$

[3]There can be partial downloads of services under RED/LED during $[1, t_1]$. In this case, $C_{\text{RL}} \leq 1.5 C_{\text{OPT}}$ during $[1, t_1]$ and the proof is a special case of the latter proof.

[4]E_i includes partial downloads and each partial download counts as one.

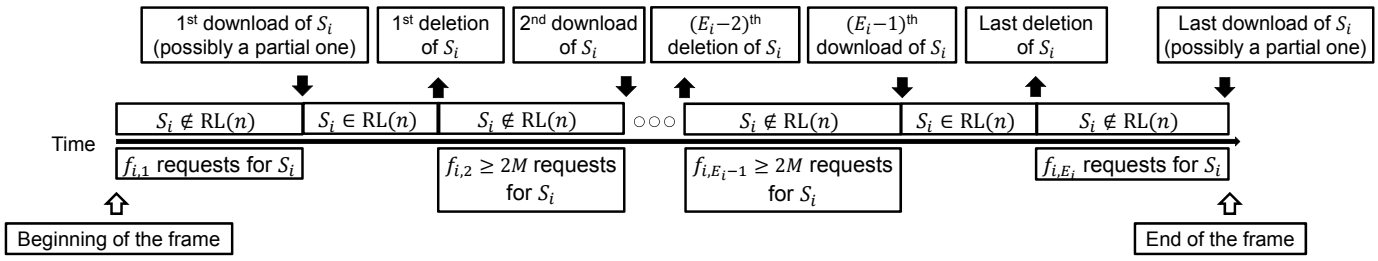


Fig. 4. An example of the downloads and deletions of S_i in a frame.

$4M$ requests to the back-end cloud, and incurs at least $f_{i,z} - 4M$ units of cost. Apply this argument for all z, and we have $C_{\text{OPT}}(g) \geq \sum_z (f_{i,z} - 4M)^+ + M$, where $x^+ := \max\{0, x\}$. The last term M is the download cost at the beginning of the frame. Let $\mathcal{S}_{\text{OPT}} := \{i \in \mathbb{Z} \mid S_i \in \text{OPT}(t_g + 1)\}$ denote the set of indices of services that are cached by OPT in the g-th frame. We now have the first bound on $C_{\text{OPT}}(g)$:

$$\begin{aligned} C_{\text{OPT}}(g) &\geq \max_{i \in \mathcal{S}_{\text{OPT}}} \sum_{z=1}^{E_i} (f_{i,z} - 4M)^+ + M \\ &\geq \max_{i \in \mathcal{S}_{\text{OPT}}} \left(\sum_{z=1}^{E_i} f_{i,z} - 4ME_i \right) + M =: B_1. \end{aligned} \tag{4}$$

Next, we consider the deletions of S_i. We have the following lemma for OPT.

Lemma 2: Suppose RED/LED deletes a service S_i at the n-th arrival and at the m-th arrival, $n < m$, and for all $n \leq l \leq m$, $\text{OPT}(l) = \text{OPT}(m)$, then OPT forwards at least $2M$ requests to the back-end cloud during $[n, m]$.

Proof: See Appendix B. ∎

By Lemma 2, for every $z > 1$, OPT forwards at least $2M$ requests between the $(z-1)$-th deletion and the z-th deletion of S_i. This gives us the second bound on $C_{\text{OPT}}(g)$:

$$\begin{aligned} C_{\text{OPT}}(g) &\geq \max_{i \in \mathcal{S}_{\text{OPT}}} 2M(D_i - 1) + M \\ &\geq \max_{i \in \mathcal{S}_{\text{OPT}}} 2ME_i - 3M =: B_2. \end{aligned} \tag{5}$$

Finally, we consider a service $S_j \notin \text{OPT}(t_g + 1)$. Let $\mathcal{S}^{\complement}_{\text{OPT}} := \{j \in \mathbb{Z} \mid S_j \in \mathbb{S} \setminus \text{OPT}(t_g + 1)\}$ denote the set of indices of services that are *not* cached by OPT in the g-th frame. Suppose there are A_j requests for S_j in this frame. OPT needs to forward all these requests to the back-end cloud, for all $S_j \notin \text{OPT}(t_g + 1)$, which gives us the third bound on $C_{\text{OPT}}(g)$:

$$C_{\text{OPT}}(g) \geq \sum_{j \in \mathcal{S}^{\complement}_{\text{OPT}}} A_j + M =: B_3. \tag{6}$$

On the other hand, RED/LED might either forward or download for each request for S_j. With the concept of partial download, each request for S_j incurs a download cost of at most $\frac{M}{2M} = 0.5$ in proportion, since at least $2M$ requests for S_j in retrospect are needed for RED to make one download of S_j. Besides, one request incurs at most one unit of forward cost. Hence, A_j requests for S_j incur at most

$$(1 + 0.5)A_j = 1.5A_j \tag{7}$$

units of cost.

Combining (3) and (7) gives us a bound on $C_{\text{RL}}(g)$:

$$C_{\text{RL}}(g) \leq \sum_{i \in \mathcal{S}_{\text{OPT}}} \left(E_i M + \sum_z f_{i,z} \right) + \sum_{j \in \mathcal{S}^{\complement}_{\text{OPT}}} 1.5A_j \tag{8}$$

We are now ready to prove Theorem 2.

Proof of Theorem 2:

$$\begin{aligned} C_{\text{RL}}(g) &\leq \sum_{i \in \mathcal{S}_{\text{OPT}}} \left(E_i M + \sum_z f_{i,z} \right) + \sum_{j \in \mathcal{S}^{\complement}_{\text{OPT}}} 1.5A_j \\ &\leq K \left[\max_{i \in \mathcal{S}_{\text{OPT}}} \left(E_i M + \sum_z f_{i,z} \right) + \sum_{j \in \mathcal{S}^{\complement}_{\text{OPT}}} 1.5A_j \right] \\ &\leq K \left[\max_{i \in \mathcal{S}_{\text{OPT}}} \left(E_i M + \sum_z f_{i,z} \right) + \sum_{j \in \mathcal{S}^{\complement}_{\text{OPT}}} 6.5A_j \right] \\ &= K(B_1 + 2.5B_2 + 6.5B_3) \leq 10KC_{\text{OPT}}(g), \end{aligned}$$

for every frame. ∎

VI. Lower Bound of the Competitive Ratio

In this section, we prove that the competitive ratio of any deterministic online policy is at least K. Since the competitive ratio of RED/LED is $10K = \Theta(K)$, this implies that RED/LED is asymptotically optimal among all deterministic online policies with respect to K, i.e. RED/LED performs at most a constant factor worse than the best deterministic online policy for large K.

Theorem 3: The competitive ratio of any deterministic online policy is at least K.

Proof: Given a deterministic online policy η, we construct a sequence of arrivals as follows: When the system starts, the first N_1 arrivals are all requests for a service $Z_1 \notin \{S_1, S_2, \ldots, S_K\}$. Recall that the edge server caches services $\{S_1, S_2, \ldots, S_K\}$ when the system starts. Therefore, the service Z_1 is not initially cached by the edge server. If η never downloads Z_1, then we choose N_1 to be arbitrarily large, and the system ends after N_1 arrivals of Z_1. In this case, OPT can download Z_1 at the first arrival, and only incurs a cost of M, while η incurs a cost of N_1. The competitive ratio of η is then $\frac{N_1}{M}$, which can be made arbitrarily large. From now on, we can assume that η downloads Z_1 after a finite number of requests for Z_1, and choose the value of N_1 so that η downloads Z_1, and deletes a service, after N_1 arrivals of Z_1. Let $Z_2 \in \{S_1, S_2, \ldots, S_K\}$ be the service that η deletes.

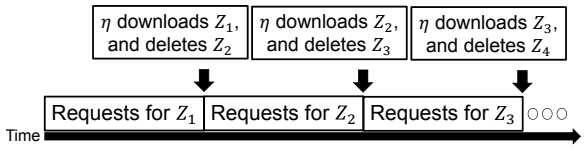


Fig. 5. An example for the proof of Theorem 3.

We construct the remaining of the sequence of arrivals iteratively: For all $k = 2, 3, \ldots, K$, there are N_k requests for service Z_k after the first $\sum_{l=1}^{k-1} N_l$ arrivals. If η never downloads Z_k, we can make the competitive ratio arbitrarily large by choosing N_k to be arbitrarily large. Therefore, we can assume that η downloads Z_k after a finite number of requests for Z_k, and choose N_k to be that number. Let $Z_{k+1} \in \{S_1, S_2, \ldots, S_K, Z_1\}$ be the service that η deletes when it downloads Z_k. The system ends after the last N_K requests for Z_K. Fig. 5 illustrates such a sequence.

By the construction of our sequence, η makes K downloads and therefore incurs a cost of at least KM. On the other hand, there are at most K different services among $\{S_1, S_2, \ldots, S_K, Z_1\}$ that have any requests in this sequence. Therefore, at least one service in $\{S_1, S_2, \ldots, S_K\}$ does not have any requests. Let Z^* be that service. An offline policy can then download Z_1 and delete Z^* when the system starts. This only incurs a cost of M, as all subsequent requests are directly served by the edge server. Therefore, the competitive ratio is at least $\frac{KM}{M} = K$. ∎

VII. Scalability of RED/LED

Theorem 3 describes a rather pessimistic result: the competitive ratio of any deterministic online policy at best increases with the capacity of the edge server. In this section, we consider improving the scalability of online policies by provisioning additional capacity to the edge server. We show that, by doubling the capacity of the edge server, our RED/LED policy achieves a constant relative cost in comparison with the optimal offline batch-download policy, which will be defined below.

A. Definitions

Definition 4 (Batch-download policy): A policy is said to be a *batch-download* policy if, whenever it makes a download, it downloads K services to the edge server, and incurs a download cost of KM.

Definition 5 (OPTb): A policy is an *optimal offline batch-download* policy and denoted by OPTb, if it minimizes the total cost of the system under any sequence of request arrivals, with full knowledge of the request sequence, among all batch-download policies.

Obviously, OPTb cannot be better than OPT. However, it is indeed OPT in the special case where $K = 1$. Moreover, as we will show in Section VII-C, there exists a polynomial time algorithm for finding OPTb, and our trace-based simulations in Section IX suggest that OPTb has similar performance as OPT in realistic settings.

We now introduce another definition of competitive ratio that takes into account the possibility of increasing the capacity of the edge server for online policies. For fair comparison under different capacities, we assume when the system starts, both edge servers cache "pseudo" services that have no request arrivals.

Definition 6: An online policy η is said to be (α, β)-OPTb-competitive, if $C_\eta \le \beta C_{\text{OPTb}}$ for every possible sequence of request arrivals, where C_η and C_{OPTb} is the total cost of η and OPTb respectively, and the capacity of the edge server under η is α times of that under OPTb.

B. Competitive Ratio of RED/LED With Additional Capacity

Theorem 4: RED/LED is $(2, 10)$-OPTb-competitive.

Throughout this section below, we assume RED/LED operates on an edge server with capacity $2K$, while OPTb operates on an edge server with capacity K.

Similar to Section V, we shall divide the arrival sequence into frames so that OPTb downloads services only at the beginning of frames. We can assume the first frame always starts before any request arrival, since otherwise it does not increase the cost of OPTb by moving its first download to the very beginning.[5] Then we compare the costs of RED/LED and OPTb in each frame. In the g-th frame, first consider a service S_i that is cached by OPTb. Recall that $\sum_{l=n}^{m} x_i(l)$ is the number of requests for S_i during $[n, m]$. We have the following lemma for OPTb.

Lemma 3: Suppose there exist integers $n < m$ such that for all $n \le l \le m$, $\text{OPTb}(l) = \text{OPTb}(n)$, and $S_i \in \text{OPTb}(l)$, but $S_i \notin \text{RL}(l)$, then the number of requests that OPTb forwards to the back-end cloud during $[n, m]$ is at least

$$K\left(\sum_{l=n}^{m} x_i(l) - 4M\right). \tag{9}$$

Proof: See Appendix C. ∎

Recall that E_i (resp. D_i) is the number of times RED/LED downloads (resp. deletes) S_i in the frame, and $f_{i,z}$ is the number of requests for S_i that RED/LED forwards to the back-end cloud within the z-th interval of S_i not being cached at the edge server in this frame. By Lemma 3, OPTb forwards at least $K(f_{i,z} - 4M)$ requests to the back-end cloud. Summing up for all z, we have the first bound on the cost of OPTb in this frame, $C_{\text{OPTb}}(g)$, as follows:

$$\begin{aligned} C_{\text{OPTb}}(g) &\ge \max_{i \in \mathcal{S}_{\text{OPTb}}} \sum_{z=1}^{E_i} [K(f_{i,z} - 4M)]^+ + KM \\ &\ge \max_{i \in \mathcal{S}_{\text{OPTb}}} \left(\sum_{z=1}^{E_i} K f_{i,z} - 4KME_i\right) + KM =: B_1', \end{aligned} \tag{10}$$

where $\mathcal{S}_{\text{OPTb}} := \{i \in \mathbb{Z} \mid S_i \in \text{OPTb}(t_g + 1)\}$ is the set of indices of services that are cached by OPTb in the g-th frame. Note that B_1' has a factor of K compared with B_1 in (4).

[5] If there is no download under OPTb, then $C_{\text{RL}} \le 1.5 C_{\text{OPTb}}$, and the proof is a special case of the later proof.

Next, consider the deletions of S_i under RED/LED. We have the following lemma:

Lemma 4: Suppose RED/LED deletes a service S_i at the n-th arrival and at the m-th arrival, $n < m$, and for all $n \le l \le m$, $\text{OPTb}(l) = \text{OPTb}(m)$, then OPTb forwards at least $2KM$ requests to the back-end cloud during $[n, m]$.

Proof: See Appendix D. ∎

By Lemma 4, between each two consecutive deletions of S_i, OPTb needs to forward at least $2KM$ requests. Therefore, we obtain the second bound on $C_{\text{OPTb}}(g)$ as follows:

$$\begin{aligned} C_{\text{OPTb}}(g) &\ge \max_{i \in \mathcal{S}_{\text{OPTb}}} 2KM(D_i - 1) + KM \\ &\ge \max_{i \in \mathcal{S}_{\text{OPTb}}} 2KME_i - 3KM =: B_2'. \end{aligned} \tag{11}$$

Again, note the factor of K in B_2' due to the fact that RED/LED with double capacity can cache K more services than OPTb.

Finally, we consider a service $S_j \notin \text{OPTb}(t_g + 1)$. Let $\mathcal{S}^{\complement}_{\text{OPTb}} := \{j \in \mathbb{Z} \mid S_j \in \mathbb{S} \setminus \text{OPTb}(t_g + 1)\}$ denote the set of indices of services that are *not* cached by OPTb in the g-th frame. Recall A_j is the number of requests for S_j in this frame. OPTb needs to forward all these requests to the back-end cloud, for all $S_j \notin \text{OPTb}$, which gives us the third bound on $C_{\text{OPTb}}(g)$:

$$C_{\text{OPTb}}(g) \ge \sum_{j \in \mathcal{S}^{\complement}_{\text{OPTb}}} A_j + KM =: B_3'.$$

On the other hand, RED/LED downloads S_j at most $\frac{A_j}{2M}$ times. S_j then at most incurs $1.5A_j$ units of cost. Adding the costs of downloading and forwarding for all S_i, we have a bound on the cost of RED/LED in the frame, $C_{\text{RL}}(g)$:

$$C_{\text{RL}}(g) \le \sum_{i \in \mathcal{S}_{\text{OPTb}}} \left(E_i M + \sum_z f_{i,z} \right) + \sum_{j \in \mathcal{S}^{\complement}_{\text{OPTb}}} 1.5A_j \tag{12}$$

We are now ready to prove Theorem 4.

Proof of Theorem 4:

$$\begin{aligned} C_{\text{RL}}(g) &\le \sum_{i \in \mathcal{S}_{\text{OPTb}}} \left(E_i M + \sum_z f_{i,z} \right) + \sum_{j \in \mathcal{S}^{\complement}_{\text{OPTb}}} 1.5A_j \\ &\le K \left[\max_{i \in \mathcal{S}_{\text{OPTb}}} \left(E_i M + \sum_z f_{i,z} \right) \right] + \sum_{j \in \mathcal{S}^{\complement}_{\text{OPTb}}} 1.5A_j \\ &\le K \left[\max_{i \in \mathcal{S}_{\text{OPTb}}} \left(E_i M + \sum_z f_{i,z} \right) \right] + \sum_{j \in \mathcal{S}^{\complement}_{\text{OPTb}}} 6.5A_j \\ &= B_1' + 2.5B_2' + 6.5B_3' \le 10 C_{\text{OPTb}}(g), \end{aligned}$$

for every frame. ∎

Theorem 4 demonstrates that RED/LED is indeed scalable, in the sense that it only needs doubled capacity to achieve a constant competitive ratio.

C. Implementation of OPTb

OPTb allows polynomial time implementation by dynamic programming. Note that once we know when OPTb makes the downloads, each download cost is a constant KM, and OPTb will download the top K popular services between two consecutive downloads to minimize the forward cost, which is then equal to the number of requests from those non-top-K services. Specifically, let $C(m)$ be the minimum total cost with batch download for a sequence of m requests. We set $C(0) = 0$. Suppose the last download occurs after the n-th arrival, then the total cost $C(m)$ is $C(m) = C(n) + KM + f(n, m)$, where $f(n, m)$ is the number of requests from the non-top-K services during $[n + 1, m]$. If there is no download at all, then $C(m) = \sum_{l=1}^{m} \mathbb{1}\{r_l \notin \{S_1, \ldots, S_K\}\}$ is the number of requests whose services are not cached in the beginning. Therefore, the dynamic programming recursion is:

$$C(m) = \min \begin{cases} \min_{0 \le n < m} C(n) + KM + f(n, m), \\ \sum_{l=1}^{m} \mathbb{1}\{r_l \notin \{S_1, \ldots, S_K\}\}. \end{cases}$$

For a sequence of N requests, it is easy to check the time complexity is at most $O(N^2(N + |\mathbb{S}| \log |\mathbb{S}|))$, where $|\mathbb{S}|$ is the total number of services in the system.

VIII. PRACTICAL ISSUES

A. Extensions to Heterogeneous Systems

Our analysis so far has assumed that all services are homogeneous. In practice, however, some services are very sensitive to delays, while others are not. Different services also require different amounts of edge-server resources and incur different download costs. We now discuss how to address these heterogeneous features.

We model the heterogeneous features as follows: Forwarding a request for S_i to the back-end cloud incurs a cost of F_i, and downloading the service S_i to the edge server incurs a cost of M_i, with $M_i \ge F_i > 0$. Each service S_i requires $W_i > 0$ units of edge-server resources, and the edge server only has K units of resources. Therefore, if the edge server caches a subset $\mathbb{T}$ of services, we then have $\sum_{i:S_i \in \mathbb{T}} W_i \le K$. Our previous analysis corresponds to the special case where $F_i \equiv 1$, $M_i \equiv M$, and $W_i \equiv 1$.

We have the following theorem for OPT in heterogeneous systems:

Theorem 5: Suppose we are given a sequence $r_1, r_2, \ldots$, and $\text{OPT}(n)$, which is the subset of services cached by OPT after the n-th arrival. If there exists an integer $m > n$, a service $S_i \notin \text{OPT}(n)$, and another service $S_j \in \text{OPT}(n)$ with $W_j \ge W_i$ such that:

$$\sum_{l=n+1}^{m} F_i x_i(l) \ge \sum_{l=n+1}^{m} F_j x_j(l) + M_i + M_j, \tag{13}$$

then OPT downloads at least one service during $[n + 1, m]$.

Proof: The proof is virtually the same as that of Theorem 1. ∎

With the intuition provided by Theorem 5, we can modify RED and LED as follows:

Definition 7: When a request $r_n = S_i \notin \text{RL}(n - 1)$ arrives, RED downloads and replicates S_i at the edge server if there

exists an integer τ and a service S_j such that for all $n-\tau \le l \le n-1$, $S_j \in \text{RL}(l)$ and $S_i \notin \text{RL}(l)$, and

$$\sum_{l=n-\tau}^{n} F_i x_i(l) \ge \sum_{l=n-\tau}^{n} F_j x_j(l) + M_i + M_j. \qquad (14)$$

Definition 8: Suppose the edge server decides to download a service at the n-th arrival. For each service S_i currently cached by the edge server, let τ_i be the smallest integer such that $\sum_{l=n-\tau_i}^{n} F_i x_i(l) \ge 2M_i$. LED sorts all cached services in descending order of τ_i, and deletes services in this order until there are enough resources to accommodate the new service.

B. Implementation and Complexity

This subsection discusses the implementation and complexity of RED/LED. We focus on the homogeneous system to simplify the notation. However, it is straightforward to extend the implementation for heterogeneous systems.

We first discuss the implementation of the retrospective download (RED) policy. For all $S_i \in \text{RL}(n-1)$ and $S_j \notin \text{RL}(n-1)$, let $\mathcal{S}_\tau := \{\tau \in \mathbb{Z}^+ \mid S_i \in \text{RL}(l), S_j \notin \text{RL}(l), \forall l \in [n-\tau, n-1]\}$, and define

$$b_{ij}(n) := \left[\max_{\tau \in \mathcal{S}_\tau} \sum_{l=n-\tau}^{n} x_j(l) - \sum_{l=n-\tau}^{n} x_i(l)\right]^+.$$

By the definition of RED, a service S_j will be downloaded at the n-th arrival if $b_{ij}(n) \ge 2M$, for some $S_i \in \text{RL}(n-1)$. Finding the value of $b_{ij}(n)$ can be transformed into the well-known maximum subarray problem as follows: Construct a sequence of integers $\{a_n\}$ such that $a_n = 1$ if $x_j(n) = 1$, $a_n = -1$ if $x_i(n) = 1$, and $a_n = 0$, otherwise. We then have

$$b_{ij}(n) = \left[\max_{\tau \in \mathcal{S}_\tau} \sum_{l=n-\tau}^{n} a_l\right]^+,$$

which can be computed easily as follows: If service S_i is downloaded at the n-th arrival, set $b_{ij}(n) = 0$, for all j. Otherwise, $b_{ij}(n) = [b_{ij}(n-1) + x_j(n) - x_i(n)]^+$.

Next, we discuss the implementation of LED. When RED/LED decides to download a service, LED needs to compute τ_i for all services S_i cached by RED/LED, where τ_i is chosen so that there are $2M$ requests for S_i in the last τ_i requests. In order to obtain τ_i, each service can maintain the arrival times of its last $2M$ requests, which can be easily done by using a queue of size $2M$ to store the arrival times of past requests.

It is straightforward to extend the above discussions for heterogeneous systems. The complete pseudocode of RED/LED for heterogeneous systems is shown in Algorithm 1. It is easy to check that the time complexity of RED/LED is $O(|\mathbb{S}|)$ per request for homogeneous systems and $O(K|\mathbb{S}|)$ per request for heterogeneous systems, where $|\mathbb{S}|$ is the total number of services in the system. The space complexity is $O(|\mathbb{S}|^2)$. Even when the number of unique services is as large as 10^4, the memory footprint of RED/LED is only about 400 MB, which is easily manageable for edge servers.

Algorithm 1 RED/LED for Heterogeneous Systems

```
b_ij ← 0, ∀i, j
n ← 0
Initialize a queue, q_i, with ⌈2M_i/F_i⌉ elements of 0, ∀i
while a new request arrives do
    Suppose the request is for service i*
    n ← n + 1
    Push n into q_{i*}, and pop out an element
    if i* ∈ RL(n − 1) then
        Serve this request at the edge server
        for j ∉ RL(n − 1) do
            b_{i*j} ← (b_{i*j} − F_{i*})^+
    else
        c ← False    // whether to cache i*
        for j ∈ RL(n − 1) do
            b_{ji*} ← b_{ji*} + F_{i*}
            if b_{ji*} ≥ M_j + M_{i*} and W_{i*} ≤ K then
                c ← True
        if c then
            τ_j ← n − (head of q_j), ∀j ∈ RL(n − 1)
            repeat
                Delete j in descending order of τ_j
            until there are enough resources for i*
            Download i*
            b_{i*j} ← 0, ∀j
            for j that has just been deleted do
                b_ij ← 0, ∀i
        else
            Forward this request to the back-end cloud
```

IX. Trace-Based Simulations

In this section, we compare the performance of RED/LED against OPTb, three other offline policies, and one online policy, all using real-world data traces.

A. Overview of the Trace

We use a data set from a Google cluster [15] to obtain the sequences of service requests.[6] This data set registers more than three million request arrivals over seven hours, and the average inter-arrival time is about 7 ms. Each request has a "ParentID" that identifies its service. In this data set, there are 9,218 unique services. The most popular service has 208,306 requests, while 5,150 services each only have one request. In all the simulations, we partition the data set into ten non-overlapping parts, and run policies on these ten parts individually. We then report the average performance of these ten parts.

B. Baseline Policies

In addition to RED/LED and OPTb, we also implement three different offline policies and one online policy. The three offline policies are derived from optimal solutions based on data caching, stochastic optimization, and dynamic programming, respectively. We describe the implementations of these policies for both homogeneous systems and heterogeneous systems.

Belady Modified. Belady's algorithm is known to be the optimal offline solution to the data caching problem in computer architecture. It minimizes cache misses by swapping in

[6] We are not aware of a public data set from deployed edge servers yet.

a new item and deleting the item which will be used in the furthest future. To adopt it in the service caching scenario, we make a small modification: Instead of always downloading a service that is not cached by the edge server, we forward the request when the next occurrence of the requested service is further in the future than those of existing edge services.[7] With this modification, our version of Belady's algorithm takes advantage of the possibility of forwarding without downloading. Belady Modified then achieves the optimal performance in the special case where $M = 1$ for homogeneous systems. For heterogeneous systems, Belady Modified keeps deleting services whose next request is furthest in the future until it has enough resources to accommodate the new service.

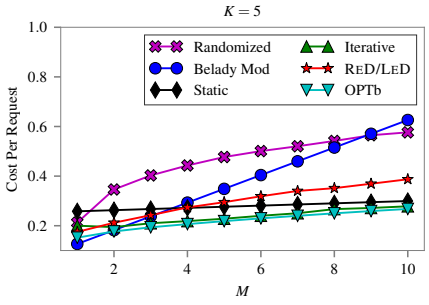
Offline Static. In homogeneous systems, Offline Static computes the frequency of all service requests and simply chooses the top K popular services to cache at the edge server. In heterogeneous systems, Offline Static caches a subset of services so that their total resource usage is no more than K, and the sum of their frequencies is maximized, which is a knapsack problem. When the arrivals of requests follow an i.i.d. stochastic process, most online policies that employ stochastic optimization will converge to Offline Static.

Offline Iterative. Given the complete trace, it is possible to compute the optimal solution, OPT, using dynamic programming. However, even for the homogeneous system, the complexity of dynamic programming is at least $O(\binom{|\mathbb{S}|}{K})$ per request. Even when K is as small as 5, our implementation finds that dynamic programming cannot be completed within a reasonable amount of time. Therefore, we instead implement the following iterative policy for homogeneous systems: Since the edge server can cache K services, we say that the edge server has K slots, numbered as $L_1, L_2, \ldots, L_K$, and each of them can cache one service. Offline Iterative algorithm finds the edge service at each of the K slots iteratively. First, it uses dynamic programming to find services cached in L_1 so as to minimize the total cost assuming the capacity is one. Given the solutions for $L_1, L_2 \ldots, L_k$, the policy then uses dynamic programming to find the services cached in L_{k+1} so that the total cost is minimized assuming the capacity is $k + 1$, and $L_1, \ldots, L_k$ are given. This policy achieves the optimal performance when $K = 1$. We only test this policy for homogeneous systems since it cannot be easily extended for heterogeneous systems.

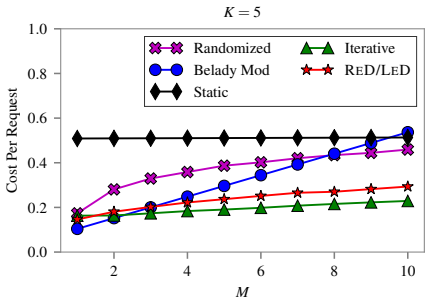
Online Randomized. We consider an online baseline policy in addition to the above offline policies. Intuitively, a reasonable policy should download more often when the download cost is small. When a request whose service is not an edge service arrives, Online Randomized downloads the new service with probability $\frac{1}{M}$, or with probability $\frac{F_i}{M_i}$ in heterogeneous systems. To accommodate the new service, Online Randomized keeps deleting edge services uniformly at random until there are enough resources available. As Online Randomized is a randomized policy, we report its average performance over 10 i.i.d. simulation runs on each part of the data set.[8]

[7]We say a service is an edge service if it is cached by the edge server.

[8]The average performance over more runs remains the same.



(a) Total cost over 10^3 requests



(b) Total cost over 10^4 requests

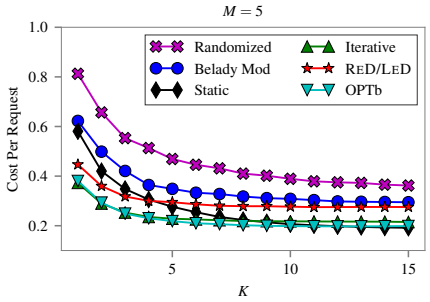
Fig. 6. Cost comparison with different download costs M.

C. Performance Evaluations for Homogeneous Systems

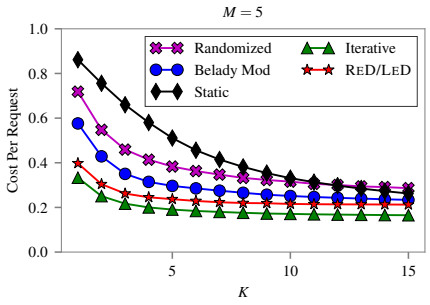
We implement all the above policies and run the algorithms with different K and M for homogeneous systems. For each pair of K and M, we calculate the total costs over 10^3 requests and over 10^4 requests. The costs, normalized by the number of requests, are compared in Fig. 6 and Fig. 7. Due to excessive running time, OPTb is skipped for the case of 10^4 requests.[9]

Fig. 6 compares the costs of the above algorithms while fixing $K = 5$. RED/LED performs very well when compared with other policies. In most settings, OPTb and Offline Iterative are slightly better than RED/LED, but the difference is very limited. We note that OPTb and Offline Iterative are very intelligent policies that require the knowledge of all future arrivals and have very high complexity. The result that our policy, being an online policy with low complexity, is only slightly worse than Offline Iterative suggests that it works well in practice. Also note that RED/LED has much better "real-world" performance than that the theoretical result guarantees since the competitive ratio is based on a worst-case analysis. Belady Modified achieves better performance than RED/LED when $M = 1$, as it is indeed the optimal policy, OPT, in such special case. However, as M becomes larger, it quickly becomes much worse than RED/LED. This highlights the difference of the service caching problem from the data caching problem. Offline Static can be better than RED/LED when we only evaluate it over 10^3 requests, but has worse performance than RED/LED when we evaluate it over 10^4 requests. With more requests, the system witnesses more variations, and therefore Offline Static becomes worse. Finally, Online Randomized performs poorly in all settings.

[9]Over seven hours are required for a single sequence in our simulation.

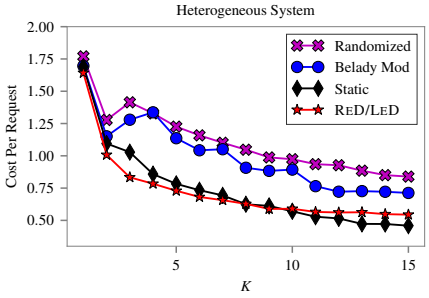


(a) Total cost over 10^3 requests

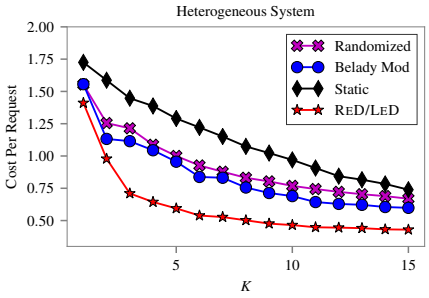


(b) Total cost over 10^4 requests

Fig. 7. Cost comparison with different K.



(a) Total cost over 10^3 requests



(b) Total cost over 10^4 requests

Fig. 8. Cost comparison in heterogeneous systems.

Fig. 7 shows the costs with different K with $M = 5$. Similar to Fig. 6, OPTb and Offline Iterative are only slightly better than RED/LED in all settings. Note that OPTb and Offline Iterative both minimize the total cost when $K = 1$. This result therefore shows that our RED/LED is close to OPT. Offline Static performs worse than our policy when we evaluate it over 10^4 requests. Both Belady Modified and Online Randomized are worse than RED/LED under all settings. Fig. 7a also exhibits the good performance of RED/LED with double capacity if we compare, for instance, the cost of RED/LED when $K = 10$ and the cost of OPTb when $K = 5$.

In addition, we also note that OPTb and Offline Iterative have very similar performance in all settings. This suggests that OPTb may be close to optimal in real-world scenarios.

D. Performance Evaluations for Heterogeneous Systems

We further evaluate the performance for heterogeneous systems. To create heterogeneity, we assign different forward costs $F_i = 1, 2, 3$, download costs $M_i = 5, 10, 15$, and resource requirements $W_i = 1, 2, 3$ to different services by their IDs. We again run the algorithms over 10^3 requests and over 10^4 requests, and calculate their total costs.

The normalized costs of different policies with different edge-server capacities K are shown in Fig. 8. We can see that RED/LED achieves the minimum cost among all policies under most settings. Although we only establish the competitiveness of RED/LED for homogeneous systems, this result suggests that RED/LED remains a desirable solution even in heterogeneous systems. Note that generally the curves are not smooth since K takes discrete values, and the non-monotonicity is due to specific request sequences, heterogeneity of costs and resource requirements, and download policies.

X. CONCLUSIONS

This paper studies online algorithms for dynamic service caching at the edge. We introduce a model that captures the limited capacity of edge servers, the unknown arrival patterns of requests, and the operational costs of edge servers, including the cost of forwarding requests to the back-end cloud and the cost of downloading new services. We propose an online policy, RED/LED, that has a small total cost under any arbitrary sequence of arrivals. We evaluate the competitive ratio of RED/LED, and prove that it is at most $10K$, where K is the capacity of the edge server. Moreover, we prove that the competitive ratio of any deterministic online policy is at least $\Theta(K)$, and therefore RED/LED is asymptotically optimal with respect to K. In addition, we show that our policy is $(2, 10)$-OPTb-competitive and thus scalable. Furthermore, RED/LED can be easily extended to heterogeneous systems and maintains a low time complexity. The performance of RED/LED is further evaluated through simulations using traces from real-world data centers. Simulation results demonstrate that our RED/LED achieves better, or similar, performance compared to many intelligent policies in all scenarios.

There are several important directions for future research: First, RED/LED is only asymptotically optimal for homogeneous systems. It is of interest to study the optimal online algorithm for heterogeneous systems. Second, RED/LED is only proved to be asymptotically optimal among deterministic online policies, and it remains an open question whether RED/LED is still optimal when randomized policies are taken into account. Third, the scalability result of RED/LED is

established by comparing it against OPTb, which is a more constrained policy than the optimal offline policy, OPT. Studying the scalability of online algorithms when compared against OPT can be important future work.

REFERENCES

[1] I.-H. Hou, T. Zhao, S. Wang, and K. Chan, "Asymptotically optimal algorithm for online reconfiguration of edge-clouds," in *ACM MobiHoc 2016*, Paderborn, Germany, Jul. 2016, pp. 291–300.

[2] R. Ravindran, X. Liu, A. Chakraborti, X. Zhang, and G. Wang, "Towards software defined ICN based edge-cloud services," in *2013 IEEE 2nd Int. Conf. Cloud Networking (CloudNet)*. IEEE, 2013, pp. 227–235.

[3] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and K. Ha, "The role of cloudlets in hostile environments," *IEEE Pervasive Computing*, vol. 12, no. 4, pp. 40–49, 2013.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st ACM Mobile Cloud Computing Workshop*. ACM, 2012, pp. 13–16.

[5] D. Willis, A. Dasgupta, and S. Banerjee, "ParaDrop: A multi-tenant platform to dynamically install third party services on wireless gateways," in *Proc. 9th ACM Workshop Mobility in the Evolving Internet Architecture (MobiArch '14)*. ACM, 2014, pp. 43–48.

[6] T. Zhao, S. Zhou, X. Guo, and Z. Niu, "Tasks scheduling and resource allocation in heterogeneous cloud for delay-bounded mobile edge computing," in *2017 IEEE Int. Conf. Communications (ICC)*, May 2017.

[7] Q. Li, W. Shi, X. Ge, and Z. Niu, "Cooperative edge caching in software-defined hyper-cellular networks," *IEEE J. Sel. Areas Commun.*, vol. PP, no. 99, 2017.

[8] J. Tadrous, A. Eryilmaz, and H. El Gamal, "Proactive content distribution for dynamic content," in *2013 IEEE Int. Symp. Information Theory (ISIT)*. IEEE, 2013, pp. 1232–1236.

[9] J. Llorca, A. M. Tulino, K. Guan, J. Esteban, M. Varvello, N. Choi, and D. Kilper, "Dynamic in-network caching for energy efficient content delivery," in *2013 Proc. IEEE INFOCOM*. IEEE, 2013, pp. 245–249.

[10] M. M. Amble, P. Parag, S. Shakkottai, and L. Ying, "Content-aware caching and traffic management in content distribution networks," in *2011 Proc. IEEE INFOCOM*, 2011.

[11] X. Qiu, H. Li, C. Wu, Z. Li, and F. Lau, "Cost-minimizing dynamic migration of content distribution services into hybrid clouds," in *2012 Proc. IEEE INFOCOM*. IEEE, 2012, pp. 2571–2575.

[12] S. Borst, V. Gupt, and A. Walid, "Distributed caching algorithms for content distribution networks," in *2010 Proc. IEEE INFOCOM*. IEEE, 2010, pp. 1–9.

[13] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *Proc. IFIP Networking*, 2015.

[14] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205–228, 2015.

[15] J. L. Hellerstein, "Google cluster data," Google research blog, Jan. 2010, posted at http://googleresearch.blogspot.com/2010/01/google-cluster-data.html.

[16] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017.

[17] A. Machen, S. Wang, K. K. Leung, B. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 140–147, Feb. 2018.

[18] K. Ha, Y. Abe, T. Eiszler, Z. Chen, W. Hu, B. Amos, R. Upadhyaya, P. Pillai, and M. Satyanarayanan, "You can teach elephants to dance: Agile VM handoff for edge computing," in *SEC '17*, 2017.

[19] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via Docker container migration," in *SEC '17*, 2017.

[20] G. Lewis, S. Echeverría, S. Simanta, B. Bradshaw, and J. Root, "Tactical cloudlets: Moving cloud computing to the edge," in *2014 IEEE Military Communications Conf. (MILCOM)*. IEEE, 2014, pp. 1440–1446.

[21] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017.

[22] J. Liu, T. Zhao, S. Zhou, Y. Cheng, and Z. Niu, "CONCERT: A cloud-based architecture for next-generation cellular systems," *IEEE Wireless Commun.*, vol. 21, no. 6, pp. 14–22, Dec. 2014.

[23] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, "Edge analytics in the Internet of Things," *IEEE Pervasive Computing*, no. 2, pp. 24–31, 2015.

[24] M. Satyanarayanan, Z. Chen, K. Ha, W. Hu, W. Richter, and P. Pillai, "Cloudlets: at the leading edge of mobile-cloud convergence," in *2014 6th Int. Conf. Mobile Computing, Applications and Services (MobiCASE)*. IEEE, 2014, pp. 1–9.

[25] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 2017.

[26] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, 2017.

[27] L. Yang, J. Cao, G. Liang, and X. Han, "Cost aware service placement and load dispatching in mobile cloud systems," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1440–1452, May 2016.

[28] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.

[29] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Commun. ACM*, vol. 28, no. 2, pp. 202–208, Feb. 1985.

[30] D. Achlioptas, M. Chrobak, and J. Noga, "Competitive analysis of randomized paging algorithms," *Theoretical Computer Science*, vol. 234, no. 1, pp. 203–218, 2000.

[31] N. Bansal, N. Buchbinder, and J. S. Naor, "A primal-dual randomized algorithm for weighted paging," *J. ACM*, vol. 59, no. 4, pp. 19:1–19:24, Aug. 2012.

APPENDIX A
PROOF OF LEMMA 1

This section provides a proof for Lemma 1. By assumptions of the lemma, $\text{OPT}(l) \neq \text{RL}(l)$, for all $n \leq l \leq m$. Therefore, there must exist some service that is in $\text{RL}(l)$ but not in $\text{OPT}(l)$. OPT needs to forward requests for these services to the back-end cloud. We will count the number of requests for these services.

Since the edge server can cache K services, we say that the edge server has K slots, numbered as L_k for $k = 1, 2, \ldots, K$. We use $L_k[r]$ to denote the r-th service cached at slot L_k during $[n, m]$. We use $d_k[r]$ to denote the time after which $L_k[r]$ is deleted and $L_k[r+1]$ is downloaded. We set $d_k[0] = n-1$, for all k. If $L_k[r]$ is the last service cached at L_k before the m-th arrival, we set $d_k[r] = m$. To simplify the notation, let $\delta_k[r] := d_k[r-1]+1$. Therefore, the service $L_k[r]$ is cached at L_k during $[\delta_k[r], d_k[r]]$ for all r.

Lemma 5: Suppose $L_k[r] = S_j$, then $\sum_{l=\delta_k[r]}^{d_k[r]} x_j(l) \geq \sum_{l=\delta_k[r]}^{d_k[r]} x_i(l) - 2M$.

Proof: We prove this by contradiction. By the assumption of Lemma 1, $S_i \notin \text{RL}(l)$, for all $\delta_k[r] \leq l \leq d_k[r]$. If $\sum_{l=\delta_k[r]}^{d_k[r]} x_j < \sum_{l=\delta_k[r]}^{d_k[r]} x_i - 2M$, then RED/LED would have downloaded S_i before the $d_k[r]$-th arrival. ■

We classify all services that are in $\text{RL}(l)$ but not in $\text{OPT}(l)$, for all $l \in [n, m]$, into two types:

Definition 9: Suppose $L_k[r] = S_j \notin \text{OPT}(n)$, then we say that S_j is of **type I** if $\sum_{l=n}^{d_k[r]} x_j(l) \geq \sum_{l=n}^{d_k[r]} x_i(l) - 4M$, and say that S_j is of **type II** if there exists some $\tau \geq n$ such that $\sum_{l=\tau}^{d_k[r]} x_j(l) \geq \sum_{l=\delta_k[r]}^{d_k[r]} x_i(l)$.

By this definition, OPT needs to forward at least $\sum_{l=n}^{d_k[r]} x_i(l) - 4M$ requests for $L_k[r]$ if it is of type I, and at least $\sum_{l=\delta_k[r]}^{d_k[r]} x_i(l)$ requests for $L_k[r]$ if it is of type II. It is possible for a service to be of both type I and type II. We first prove that a service $L_k[r] \notin \text{OPT}(n)$ is of either type I or type II.

Lemma 6: Suppose $L_k[r] = S_j \notin \text{OPT}(n)$, then S_j is of either type I or type II.

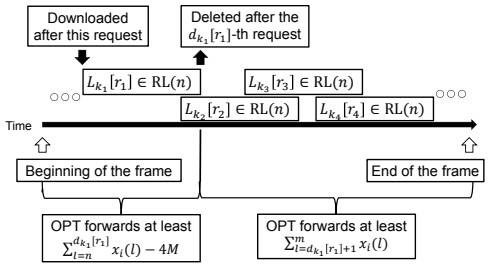


Fig. 9. An example for the proof of Lemma 1.

Proof: If $r = 1$, then $\delta_k[r] = n$, and we have $\sum_{l=n}^{d_k[r]} x_j(l) \geq \sum_{l=n}^{d_k[r]} x_i(l) - 2M > \sum_{l=n}^{d_k[r]} x_i(l) - 4M$ by Lemma 5. In this case, S_j is of type I.

Next, we consider the case $r > 1$. S_j is downloaded by RED/LED after the $(d_k[r-1])$-th arrival. By the design of RED/LED, there exists a service S_{j^*} and τ such that $S_{j^*} \in \text{RL}(l)$, for all $\tau \leq l \leq d_k[r-1]$, and

$$\sum_{l=\tau}^{d_k[r-1]} x_j(l) \geq \sum_{l=\tau}^{d_k[r-1]} x_{j^*}(l) + 2M. \tag{15}$$

If $\tau \geq n$, then we have

$$\begin{aligned}\sum_{l=\tau}^{d_k[r]} x_j(l) &= \sum_{l=\tau}^{d_k[r-1]} x_j(l) + \sum_{l=\delta_k[r]}^{d_k[r]} x_j(l) \\ &\geq \sum_{l=\tau}^{d_k[r-1]} x_{j^*}(l) + 2M + \sum_{l=\delta_k[r]}^{d_k[r]} x_i(l) - 2M \\ &\geq \sum_{l=\delta_k[r]}^{d_k[r]} x_i(l),\end{aligned}$$

and S_j is of type II. On the other hand, if $\tau < n$, then we have

$$\sum_{l=\tau}^{n-1} x_j(l) < \sum_{l=\tau}^{n-1} x_{j^*}(l) + 2M, \tag{16}$$

or S_j would have been downloaded earlier. Combining (15) and (16) yields $\sum_{l=n}^{d_k[r-1]} x_j(l) \geq \sum_{l=n}^{d_k[r-1]} x_{j^*}(l)$. Therefore,

$$\begin{aligned}\sum_{l=n}^{d_k[r]} x_j(l) &= \sum_{l=n}^{d_k[r-1]} x_j(l) + \sum_{l=\delta_k[r]}^{d_k[r]} x_j(l) \\ &\geq \sum_{l=n}^{d_k[r-1]} x_{j^*}(l) + \sum_{l=\delta_k[r]}^{d_k[r]} x_i(l) - 2M \\ &\geq \sum_{l=\delta_k[r]}^{d_k[r]} x_i(l) - 4M,\end{aligned}$$

and S_j is of type I. ∎

We are now ready to prove Lemma 1.

Proof of Lemma 1: Let $L_{k_1}[r_1]$ be the type I service with the largest $d_k[r]$. Since $\text{OPT}(l) \neq \text{RL}(l), \forall l \in [d_{k_1}[r_1]+1, m]$, we can find a set of type II services $\{L_{k_2}[r_2], L_{k_3}[r_3] \dots\}$ such that the union of $[d_{k_j}[r_j-1]+1, d_{k_j}[r_j]]$ covers $[d_{k_1}[r_1]+1, m]$. Fig. 9 illustrates an example of finding $L_{k_1}[r_1], L_{k_2}[r_2], \dots$. By the definition of type II service, the total number of requests that OPT needs to forward for these services is at least $\sum_{l=d_{k_1}[r_1]+1}^{m} x_i(l)$. Also, OPT needs to forward at least $\sum_{l=n}^{d_{k_1}[r_1]} x_i(l) - 4M$ requests for $L_{k_1}[r_1]$. Therefore, in total, OPT needs to forward at least $\sum_{l=n}^{m} x_i(l) - 4M$ requests. ∎

APPENDIX B
PROOF OF LEMMA 2

Proof of Lemma 2: By the first condition of RED, there are at least $2M$ requests for S_i during $[n, m]$. Otherwise, S_i cannot be downloaded during $[n, m]$, and therefore cannot be deleted at the m-th arrival.

When S_i is to be deleted at the m-th arrival, it must have the largest τ_i among all services currently cached by RED/LED, where τ_i is defined in Def. 3. Since there are at least $2M$ requests for S_i during $[n, m]$, all services in $\text{RL}(m)$ have at least $2M$ requests during $[n, m]$.

First, consider the case $\text{RL}(m-1) \neq \text{OPT}(m-1)$. There must exist a service S_j such that $S_j \in \text{RL}(m-1)$, but $S_j \notin \text{OPT}(m-1)$. OPT needs to forward all requests for S_j during $[n, m]$, and there are at least $2M$ of them.

Next, consider the case $\text{RL}(m-1) = \text{OPT}(m-1)$. At the m-th arrival, RED/LED deletes S_i in order to download another service $S_j \notin \text{RL}(m-1) = \text{OPT}(m-1)$. By the design of RED/LED, there exists τ and a service $S_{i^*} \in \text{RL}(m-1)$ such that the conditions in Def. 2 are satisfied. In particular, $\sum_{l=m-\tau}^{m} x_j(l) \geq \sum_{l=m-\tau}^{m} x_{i^*}(l) + 2M$. If $m - \tau \geq n$, then there are at least $2M$ requests for S_j during $[n, m]$, and OPT needs to forward all of them. On the other hand, consider the case $m - \tau < n$. Since RED/LED does not download S_j until the m-th arrival, we have $\sum_{l=m-\tau}^{n-1} x_j(l) < \sum_{l=m-\tau}^{n-1} x_{i^*}(l) + 2M$, and therefore $\sum_{l=n}^{m} x_j(l) \geq \sum_{l=n}^{m} x_{i^*}(l) \geq 2M$. OPT still needs to forward at least $2M$ requests. ∎

APPENDIX C
PROOF OF LEMMA 3

Note that Lemma 5 still holds. For RED/LED with double capacity, there are at least K services that are in $\text{RL}(l)$ but not in $\text{OPTb}(l)$ for any $n \leq l \leq m$. These services are either of type I or type II:

Definition 10: Suppose $L_k[r] = S_j \notin \text{OPTb}(n)$, then we say that S_j is of **type I** if $\sum_{l=n}^{d_k[r]} x_j(l) \geq \sum_{l=n}^{d_k[r]} x_i(l) - 4M$, and say that S_j is of **type II** if there exists some $\tau \geq n$ such that $\sum_{l=\tau}^{d_k[r]} x_j(l) \geq \sum_{l=\delta_k[r]}^{d_k[r]} x_i(l)$.

Lemma 7: Suppose $L_k[r] = S_j \notin \text{OPTb}(n)$, then S_j is of either type I or type II.

Proof: The proof is virtually the same as the proof of Lemma 6. ∎

We are now ready to prove Lemma 3.

Proof of Lemma 3: Recall that we can find K services that are not cached by OPTb but by RED/LED after each arrival during $[n, m]$, and they are either of type I or type II. Since the exact location of a service in the $2K$ slots does not affect the policy, we can think of the K services as packed in the first K rows out of the $2K$ slots during $[n, m]$, and assume the unchanged services between consecutive arrivals

stay in their row. Because there will be at most one download after each request arrival, the sets of K services differ at most by one element between any consecutive arrivals. If a service in these K rows is deleted, we relabel the service so that it appears to be a new service afterwards. Note that the relabeling does not change the decision or the cost of RED/LED and OPTb.

For each of the K rows, we can count the number of requests that OPTb needs to forward for the services in the row in a similar fashion as in the proof of Lemma 1. Let $L_{k_1}[r_1]$ be the type I service with the largest $d_k[r]$. The rest services in this row are of type II. By the definition of type II service, the total number of requests that OPTb needs to forward for these services is at least $\sum_{l=d_{k_1}[r_1]+1}^{m} x_i(l)$. Besides, OPTb needs to forward at least $\sum_{l=n}^{d_{k_1}[r_1]} x_i(l) - 4M$ requests for $L_{k_1}[r_1]$. Therefore, in total, OPTb needs to forward at least $\sum_{l=n}^{m} x_i(l) - 4M$ requests. Note the result holds for extreme cases with only type I services or only type II services in one row.

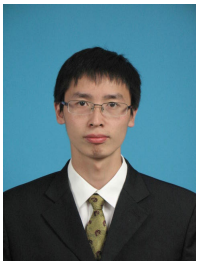
With service relabeling, the services in each row will not appear in any other row, and thus there is no duplicate in summing up the cost of all K rows. Therefore, in total, OPTb needs to forward at least $K(\sum_{l=n}^{m} x_i(l) - 4M)$ requests. ■

APPENDIX D
PROOF OF LEMMA 4

Proof of Lemma 4: By the first condition of RED as in Def. 2, there are at least $2M$ requests for S_i during $[n, m]$. Otherwise, S_i cannot be downloaded during $[n, m]$, and therefore cannot be deleted at the m-th arrival.

When S_i is to be deleted at the m-th arrival, it must have the largest τ_i among all services currently cached by RED/LED, where τ_i is defined in Def. 3. Since there are at least $2M$ requests for S_i during $[n, m]$, all services remaining in RL(m) have at least $2M$ requests during $[n, m]$.

Now consider RL$(m-1)$. There must exist at least K services $S_{j_1}, S_{j_2}, \ldots, S_{j_K}$ such that $S_{j_k} \in$ RL$(m-1)$, but $S_{j_k} \notin$ OPTb$(m-1)$, for all $k = 1, 2, \ldots, K$. OPTb needs to forward all requests for these K services during $[n, m]$, and there are at least $2KM$ of them. ■

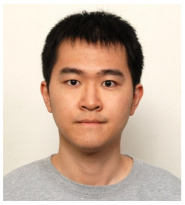


Tao Zhao received his B.Eng. and M.Sc. degrees from Tsinghua University, Beijing, China, in 2012 and 2015, respectively. He is currently a PhD student in Department of Electrical & Computer Engineering, Texas A&M University, College Station, Texas, United States. His current research interests include wireless networks, cloud-based systems, and networked systems. He received the Best Student Paper Award in WiOpt 2017.

I-Hong Hou (S'10–M'12) received the B.S. in Electrical Engineering from National Taiwan University in 2004, and his M.S. and Ph.D. in Computer Science from University of Illinois, Urbana-Champaign in 2008 and 2011, respectively.

In 2012, he joined the department of Electrical and Computer Engineering at the Texas A&M University, where he is currently an assistant professor. His research interests include wireless networks, wireless sensor networks, real-time systems, distributed systems, and vehicular ad hoc networks.

Dr. Hou received the Best Paper Award in ACM MobiHoc 2017, the Best Student Paper Award in WiOpt 2017, and the C.W. Gear Outstanding Graduate Student Award from the University of Illinois at Urbana-Champaign.



Shiqiang Wang received the Ph.D. degree from Imperial College London, United Kingdom, in 2015. Before that, he received the M.Eng. and B.Eng. degrees from Northeastern University, China, in 2011 and 2009, respectively. He joined IBM T. J. Watson Research Center in 2016 as a Research Staff Member, where he was also a Graduate-Level Co-op in the summers of 2014 and 2013. In the fall of 2012, he was at NEC Laboratories Europe, Heidelberg, Germany. His current research focuses on theoretical and practical aspects of mobile edge computing, cloud computing, and machine learning. He serves as an associate editor for IEEE Access, and has served as a technical program committee (TPC) member or reviewer for a number of international conferences and journals. He received the 2015 Best Student Paper Award of the Network and Information Sciences International Technology Alliance (ITA).

Kevin Chan is research scientist with the Computational and Information Sciences Directorate at the U.S. Army Research Laboratory (Adelphi, MD). His research interests are in network science and distributed processing. He received the Best Paper Award at the 2017 Asian Internet Engineering Conference and 2013 Conference on Behavior Representation in Modeling and Simulation; he also received the NATO Scientific Achievement Award in 2014 and 2018. He is the co-editor for the IEEE Communications Magazine Special Issue on Military Communications. He has also served on the TPC for networking and communications conferences, including IEEE DCoSS, SECON, MILCOM, and VTC. Prior to ARL, he received the PhD in Electrical and Computer Engineering and the MSECE from Georgia Institute of Technology, Atlanta, GA. He also received the BS in ECE/Engineering & Public Policy from Carnegie Mellon University, Pittsburgh, PA.