Delft University of Technology

Masters Thesis

Caching for mobile users in edge networks

Author:
N.Q. Belzer

Supervisor:
Dr. J.S. Rellermeyer

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Distributed Systems Group
Department of Software Technology

September 27, 2021

Abstract

Mobile networks deal with an increasing portion of the IP Traffic due to the significant growth in the number of mobile devices and the accompanied lifestyle. A large fraction of this IP traffic is spent on duplicate transfers for the same resources. Previous work has shown that a Content Delivery Network (CDN) based on edge-nodes can reduce redundant backhaul traffic by storing popular content closer to the user. It also shows that a reduced user population per node, as in edge-based caching systems, can have a significant negative impact on caching performance. This effect is attributed to a reduced view on global content popularity. In this work we first create and evaluate a simulator for resource requests that is used to evaluate different caching strategies in an edge network. Our simulation confirms the findings of previous work and inspire three different caching strategies: *Cooperative-LRU*, *User Profiles*, and *Hybrid with Federated*. These strategies include mobility information to help alleviate the reduced knowledge on content popularity and help nodes work together more efficiently. Our results show that we are successful in decreasing the impact of the reduced population using the Cooperative LRU strategy and Profiles strategy. We then improve upon that performance by using a Hybrid strategy of Federated nodes and one of the mobility strategies.

Acknowledgements

Before the start of this thesis I would like to acknowledge several people for their help and support that made this thesis possible.

First and foremost I want to thank my supervisor, Dr. J.S. Rellermeyer, for the guidance and ideas that contributed to this document. He provided valuable insights in my writing, experiments, and general academic skills that I will be able to take with me.

I would also like to thank my family and friends who have been nothing but supportive during the entire period of my master studies and especially during the thesis.

Contents

SECTION 1

Introduction

The IP traffic on mobile networks is seeing a compound annual growth of 46% over the period 2017-2022, compared to the compound annual growth of 26% for the total sum of IP Traffic [14]. By 2022 the mobile network will account for 20% of the total IP traffic, or 78 exabytes of traffic per month, while fixed connections (including WiFi connections to mobile devices) account for the remaining 80%. This significantly larger proportion of, the already growing, IP Traffic is attributed to the significant growth of mobile devices and lifestyles that come with them [14]. For example entertainment-, maps-, and the increasing amount of smart devices-in vehicles that all rely on the cellular network due to their mobility [22].

During the same period, the average mobile internet speed on smartphones is expected to increase to 44 Mbps by 2023 globally, up from 13 Mbps in 2017, while 5G deployments will reach an average speed around 575 Mbps [10]. Video content will make use of these higher speeds, to serve higher quality image, as it is expected to take up 61 exabyte out of the total of 77 exabyte of traffic volume per month in 2022 on the mobile network alone [14].

The rise in mobile IP traffic poses serious challenges for the mobile network [34, 15]. Extending the network capacity comes with an increased capital expenditure and operating expenditure while at the same time the average revenue per user decreases as they expect more data at lower prices [34]. In addition, due to the centralised architecture of mobile network, requests follow a long path over the backhaul of the mobile network to the core of the network before reaching the internet [31, 32]. This uses a large amount of network resources, introduces delay, and can potentially become a bottleneck during high bandwidth scenarios [34, 35, 32, 29].

Edge Computing is one of the proposed technologies that will move computing, networking, and storage capabilities closer to the end user [18]. In addition to the traditional capabilities, edge computing also offers new ones due to their locality, for example: geographical awareness and shorter response times [18, 19]. For readers interested in edge computing we redirect you to the survey by Li et al. [18].

Several works [22, 32, 33, 34] propose Edge Caching, or an Edge-Content Delivery Network (CDN), as one of the solutions to deal with the increased size of IP traffic served by mobile networks. Caching poses a cost effective solution to otherwise expensive network upgrades to optimise the usage of the network capacity [34], and therefore the energy efficiency [32]. Edge caching, due to its distributed nature, can also be more resilient to node failures, as traffic can be redirected to neighbouring nodes instead [26]. Additionally an edge-based-CDN, when deployed within the mobile network, can be given access to mobile-subscriber data such as device, location and connectivity data [37].

A Content Delivery Network works particularly well within mobile networks, because

a large fraction of the traffic volume is made up of duplicated requests for popular content [32, 8, 15]. An analysis on HTTP flows from a large cellular ISP in South Korea in 2012 shows that web caching could save up to 27.7% of the down-link bandwidth [33] for their specific capture. The work of [12] analysed a billion HTTP requests on a 3G network retrieved over two days in 2010. They found that 33.4% of the content could be served from a cache deployed within the core network.

In addition to reducing backhaul traffic, an edge-based CDN would improve response times (for stored content) for end users and improve the efficiency of the backhaul within the operators' network. This increase in efficiency can translate into additional savings in energy usage and therefore also energy expenditure [32].

Content Delivery Networks position themselves between the user and content [20]. This allows them to divide the HTTPS traffic into a front- (*user-to-cdn*) and back-end (*cdn-to-origin*) connection where both connections can be encrypted [20]. Because of this position between the encrypted channels a CDN node is able read and serve requested resources. This is important as nowadays the majority of the network requests are made using HTTPS[1] which makes every request unique [24].

To position the CDN in between the user and content provider there is a need for the content provider to forward traffic to the CDN. This is typically done through DNS redirection [25] to select a specific node based on a set condition (like proximity to the user, or known location of the requested content). Because of this explicit requirement the number of resources that we can serve from an edge-CDN are limited.

Caching solutions, specifically for mobile networks, also need to be able to deal with mobile users, for example by storing content on multiple nodes [19]. Edge nodes are by definition more restricted in terms of computing power, storage capacity, and network capabilities compared to a large datacenters used in cloud computing or the mobile network core. This poses additional challenges for cache systems, as for example storage capacity is related to the performance of a cache [19]. Additionally several works have found that a steep drop-off in performance exists for reactive cache nodes that serve a limited population size [11, 17, 21].

Certain edge solutions already exist, for example Amazon Cloudfront, is a CDN with more than 225 edge locations, spread out over the globe, including within the networks of Telecom providers[2]. It is, however, not mentioned whether any cooperation happens between the different edge locations.

In our work, while we focus on a mobile-CDN (an edge-CDN within the mobile operator network), we do not specify any specific implementation constraints that limit our findings to a mobile network operated CDN. There are, however, several upcoming technologies within mobile networks such as Multi-Access Edge Computing and Network Function Virtualisation [13, 16] that we believe would provide an ideal environment for the development of an edge-CDN. In addition to current plans this would increase the return on investment for mobile operators on the transition to enable multi-access edge

[1] Over 84% of traffic is encrypted with HTTPS in April 2021 according to letsencrypt.org/stats/.

[2] Amazon Cloudfront (aws.amazon.com/cloudfront) has more than 225 points of presence in 90 cities (Accessed on 14 Sep 2021).

computing and network function virtualisation.

1.1. The Journey

When presented with the final work it might seem like the path taken was a straightforward one. Yet at many points during the project the path was not so clear. Many detours have been taken to arrive at the point I stand now. In this section I would like to describe how we decided to focus on mobility in edge caching. I hope this section might encourage others on their own journey.

The initial proposal for my thesis discussed a topic that had been discussed between my supervisor, Dr. J.S. Rellermeyer, and Paul Voskuilen from the AMS Institute[3]. Due to a large amount of datacenters in the region Noord-Holland[4] there was an interest in improving their energy efficiency. In our initial talks we discussed how 5G technology might be able to help improve the situation.

After a few weeks of reading up on 5G technology I was unable to find a link between 5G technology and potential energy savings within datacenters. Yet while this path showed a dead end, it did light up a different one. During my readings on 5G technology I was introduced to Multi-Access Edge Computing (MEC), defined by a set of standards written by ETSI. MEC is supposed to enable computing, storage, and network capabilities at the edge (much closer to the user than cloud computing) and could be incorporated within the mobile network. This means that MEC could be an enabler for many of the promises of 5G such as ultra-low latencies and high bandwidth transfers [13].

The field of edge computing and specifically MEC is filled with proposals for latency-sensitive- and IoT-applications. I was not convinced I could make a significant contribution in these areas. Yet in one of the meetings with my supervisor the discussion on latency-sensitive applications led to the idea of edge-caching.

From there on out I spent the rest of my time on this path of edge-caching. Several detours were made trying to find a recent dataset to use, trying to set up a simulated platform to deploy MEC applications, and experimenting with information-centric networking. In the end most of this work is not visible in the final result, yet it is part of the journey.

1.2. Problem Statement

Local caching solutions, by design, serve a smaller population of connected users. This limits how much information is available to estimate the popularity distribution and can have an impact on cache performance.

In this thesis work we try to mitigate the effects of this limited popularity by different

[3]Paul Voskuilen is *Program Developer for Urban Energy Research & Valorization* at the AMS Institute.

[4]According to noord-holland.nl there are 57 datacenters in the region Noord-Holland (retrieved 16 September 2021).

methods of cooperation between local cache nodes using mobility information (from the mobile network). We do this by testing different strategies in simulated scenarios. Over the course of the work we will answer the following research questions:

RQ1 How can we create a simulated environment for cache performance testing? (answered in section 3)

RQ2 What is the effect of a limited scope on content popularity for a local cache node? (answered in section 4)

RQ3 How can cooperation with neighbouring nodes decrease the effects of a limited population? (answered in section 6)

RQ4 What are the trade-offs and limitations of the proposed strategies? (answered in section 6)

1.3. Contributions and Thesis Outline

Based on our research questions the main contributions of this work are the following:

- Propose and open-source a simulation for testing edge caching strategies, given no available representative request trace or dataset.
- Identify and verify the main bottlenecks in performance of edge-cache nodes.
- Propose and analyse the characteristics of different strategies that avoid these bottlenecks.
- Analyse the differences of the proposed strategies for three different popularity distributions based on previous work.

The rest of the work is outlined as follows: In section 2 we focus on related work in the field of edge caching. section 3 describe how we collected a dataset and used it to set up a user simulation able to simulate both resource requests and user movement. We then verify the effects of a limited population in section 4 as described in previous work for our simulation. Based on our findings we propose to test three caching strategies that use mobility information in section 5. We evaluate these strategies using our simulation in section 6 and conclude our work in section 7.

SECTION 2

Related Work

Previous work has found different ways to decrease the network load on the mobile network by different caching mechanism or setups. Here we discuss several different works from the last decade.

Xie et al. [34] introduce a (mobile) network wide caching solution using a proxy and tracker to help direct requests to the appropriate cache server. Most interesting to our own work, however, is the proposed use of data forwarding tunnels to support seamless mobility. As each base station is connected to a proxy instance it allows for the setup of indirect forwarding tunnels when the user moves. Whenever a user handover (move between base stations) occurs with requests in-flight a tunnel is set up between the original and target proxy. Any responses from the in-flight responses arrive at the original proxy and are forwarded to the new target proxy. From there the response is able to reach the user as if nothing changed.

Similar behaviour was implemented in the prototype by Giust et al. [15] —designed to provide extremely mobility-enabled local caches, with access to the internet as well as the mobile core, in 5G networks— using an IPv6-in-IPv6 tunnel to deliver in-transit packages through a new access point (even when switching from the mobile network to WiFi). In their results they show how their prototype is able to successfully deal with the handover of the user by redirecting the in-flight responses as soon as the user connects to a new node.

Wang et al. [32] propose an edge caching scheme based on content-centric networking. Content-centric networking works, in contrast to IP traffic, by giving each piece of content a unique address (instead of the server address). Interest for a resource is shared through pending interest tables when the resource is not available locally. The authors analyse three caching setups that expand on themselves: they start with global mobile network core caching, expand it with base station caching, and in their final setup include device-to-device caching. A cooperative system is implemented where popular content can be stored on different hierarchical levels as to avoid duplication at every base station. Their results show how content-centric caching outperforms both base station and core network caching in traffic load reduction and content access delay. The work of Leconte et al. [17] is one of the first works to address and propose a solution to the issue of cache performance degradation seen at small user populations. As the number of users per node decreases the cache will not have enough information to estimate content popularity, resulting in a steep drop-off in cache performance [17]. The authors propose to alleviate this issue by using partial caching combined with a form of local and global popularity learning through an *age-based threshold (ABT)*. In partial caching only the first part of the content is stored and on hit the rest is retrieved. This improves the hit-rate performance and is mostly used for multimedia streaming. The age based threshold is a cache metric that acts as a *cache-threshold-policy*. It can be learned on both a single cache as well as globally. The

threshold is based on the age of the content (since it was first inserted). The authors show global learning is faster, but that for the popularity of local content, local learning provides better performance. In their results an ABT-score-based pre-fetching approach performs better compared to a reactive ABT-score-gated.

Another approach to edge caching includes user devices. Anjum et al. [1] specifically propose the usage of device-to-device communication to bootstrap the streaming process of starting fragments for popular videos. Similarly to Leconte et al. [17] a partial cache is used to save only the initial fragments of the videos. This allows for more videos to be stored on user devices. Additionally they support their network using centralised tracker-servers that keep track of which fragments are stored on which end devices. In their simulation they find that the hit ratio is highly dependent on the size of the initial chunk of data and the library size. In their conclusion the authors mention how "in practice users are unwilling to share their data over the D2D link due to limited energy and storage resources [on their devices]" [1].

In all these different works we have not seen any combination of caching strategies and information available from the mobile network. We take three lessons from the previous work discussed:

- Caching at the mobile edge can save significant chunks of backhaul capacity [22, 32, 33, 34],
- Learning local content popularity is best done locally [1],
- However, reactive caching strategies are negatively impacted by the reduced population size that we will see in edge deployments [11, 17, 21].

In addition we recommend the survey by Li et al. [19] for interested readers.

SECTION 3

Simulation

There are different performance metrics for an edge-based cache as outlined by Wang et al. [30], which include: *Throughput*, *Backhaul cost*, *Power Consumption*, *Network Delay*, and *Hit rate*. In this work we focus specifically on improving hit ratios for edge caches and reducing the backhaul transmissions (or backhaul cost).

To capture these two performance metrics for a specific caching strategy we need to evaluate it in a controlled environment where we control both the user(s) and origin server. The origin server will host a large number of files (at minimum more than the cache is able to store) while the user will request these files from the cache. This process is visualised in Figure 3.1.

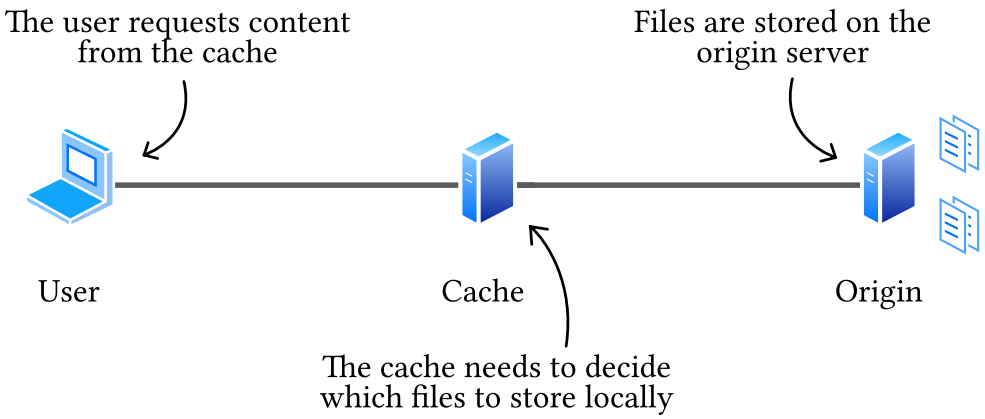


FIGURE 3.1: *A setup to evaluate a cache.*

In this setup the amount of requests served by the cache can be calculated by removing the amount of requests received by origin r_{origin} from the total amount of requests made by the user r_{total}. The requests received by origin are requests that the cache alone could not satisfy, for example because the requested content was not stored locally, and thus needed to be forwarded to origin. The fraction of requests served by the cache, also hit ratio, can be calculated as follows: $\frac{r_{total}-r_{origin}}{r_{total}}$.

To calculate the amount of bandwidth saved by the cache we can perform a very similar calculation. By removing the amount of bytes served by origin b_{origin} from the total bytes received by the user b_{total} we are left with the number of bytes served by the cache. The backhaul savings, or the fraction of bytes served by the cache, is then calculated as $\frac{b_{received}-b_{origin}}{b_{received}}$.

We define the order of the requests made by the user as a *request trace*. If the same request trace is used for a different cache strategy we are able to directly compare the two strategies in the context of that request trace. Caches should be tested on the type of request traces that closely resemble the environment in which the cache will run for the best approximation of the performance.

In addition to comparing the performance of caching strategies, we make use of Belady's MIN algorithm [6] to calculate a theoretical optimum performance for a given request trace.

To evaluate caching strategies for an edge-based cache we need to use a request trace that represents real-world requests. A suitable real world location to record or retrieve such a request trace from would be an in-use CDN node or multiple. Not only does it contain the type of requests that a cache is likely to serve, it is also able to view these requests while decrypted (in contrast to the largely encrypted web[1]).

Unfortunately no publicly available dataset (request traces) exists today that matches these criteria. Previous work has used either non-public datasets [22, 38] or used a simulation based on characteristics such as a Zipf-like popularity distribution [23, 1, 4, 9, 11, 34, 39, 36]. Therefore, as we were not able to get access to a non-public dataset, nor able to collect our own (due to the sensitive nature of the content and limited access to network resources), we decided to build a simulation instead. Our simulation is able to generate requests based on a set of resources from a dataset. We collected an appropriate dataset that contains a set of webpages and the required resources to load the page. We assume that the content retrieved in our dataset is all non-dynamic and therefore cachable content.

The upside of using a simulation is that we are able to generate and manipulate our request traces to create different scenarios, something that would not have been possible with a fixed dataset. In addition to generating different scenarios we can create different traces for the same scenarios (with the same characteristics) to improve our confidence in the captured metrics.

In this chapter we explain how we set up a simulation capable of generating request traces to be used for the evaluation of caching strategies. We evaluate three different generators that could be used to model user behaviour and two in three different configurations that will be used throughout the rest of the work.

3.1. Dataset

We want our dataset to contain a realistic distribution of file sizes that would be served by a caching solution. To collect a dataset, while keeping in mind the bandwidth distribution in the real world, we decided to collect resources from the top 25 most visited websites according to the Alexa Top 100[5] and crawl to the links they provide. For each website we crawled up to a depth of 3 through their outgoing links and recorded every page, the resources on the page, and the outgoing links. We decided on the usage of the top 25

[5]Top 25 sites retrieved from alexa.com/topsites in June 2021.

Table 3.1: *Dataset Overview*

Number of domains	16.539
Number of pages	182.932
Number of resources	1.249.511
Dataset size	84GB
Average resource size	67KB
Average pages per domain	11
Average resources per page	6
Number of scripts	37.860
Number of stylesheets	19.867
Number of images	838.297
Number of videos	375

Note: Statistics on the dataset, collected on 2021-06-17.

and depth of 3, based on experimentation, to limit the size of the dataset. The outgoing links are extracted from `<a>` tags in the HTML while the resources are extracted from a subset of HTML tags like `<img>`, `<video>`, `<audio>`, `<link>`, and `<script>`.

We split the data by this crawler into a *site-graph* and *resource-dictionary*. In the site graph each vertex represents a web page and the edges between them outgoing links. The edges are considered bi-directional as a visiting user could go back in their visiting history. Every vertex is identified by the URL of the web page it represents. The *resource-dictionary* is a mapping of the web page URL to the resources required to fully load that page.

A second tool is used to fetch all the content stored in the *resource-dictionary*, it stores each resource using an identifier made up by the domain and path of the resource. During the download process we filter out any resources that cannot be retrieved from the resource-dictionary and site-graph.

Because of the lack of JavaScript evaluation on visited websites by our scraper, we are not able to simulate multimedia streaming. Even though multimedia streaming is responsible for a large chunk of the bandwidth in mobile networks [14]. This is a concession, we deemed worth making, to keep the scope of our work on user mobility.

3.1.1 Dataset Analysis

Our crawler and download tool executed on 2021-06-17 with the top 25 sites, as denoted by the Alexa Top 100 that day, and found a total of 1.249.511 resources across 182.932 different pages that together form a dataset with a size of 84GB. An overview of the dataset can be found in Table 3.1 while a distribution of the resource sizes can be found in Figure 3.2. We find that the majority of resources (about `90%`) has a size of `200KiB` or less.

The dataset mostly contains images, accounting for `67%` of the number of resources

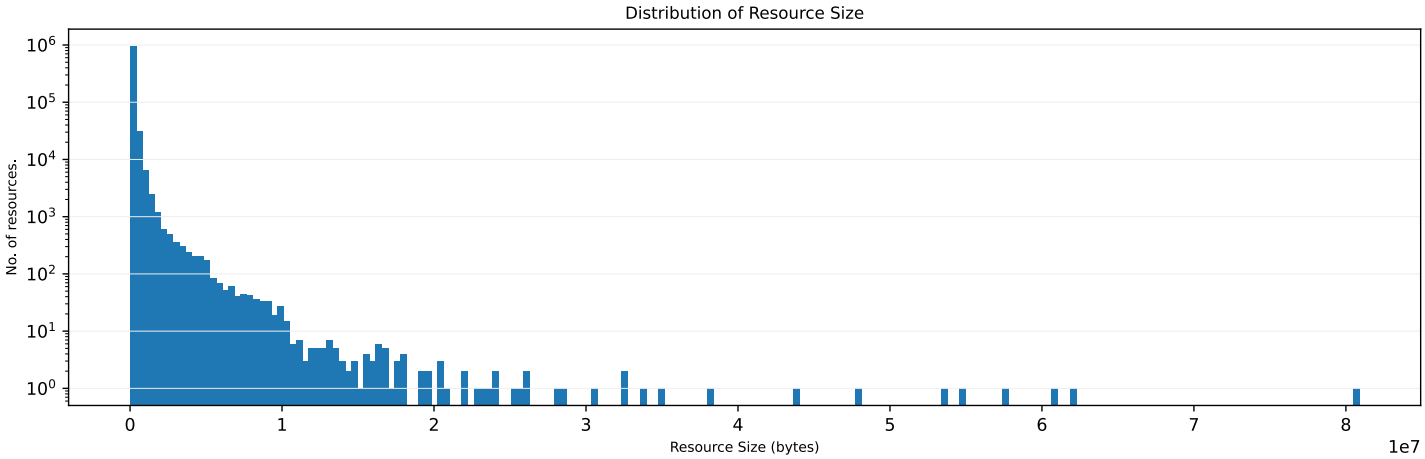


FIGURE 3.2: *The distribution of resource shown as a histogram with the Byte size on the x-axis and the number of items in each bin on a log scaled x-axis.*

and about 55% of the dataset size. Due to the low number of videos, the dataset does not match the bandwidth spent on downloading or streaming of video content as described by Cisco [10]. One of the main reasons for this is that video playback on most sites, like YouTube, is initiated by evaluating the included JavaScript resources, something that our scraper tool is not capable of.

We visualised additional characteristics of the dataset in two plots in Figure 3.3. In the first plot we split the size distributions up into a common set of different file types. The different file types are based on a grouping on the file extension. While mostly based on a list of File Formats on Wikipedia[6] some categories such as scripts, HTML, and stylesheets have been extracted because of their important role in web pages. The second plot displays the relation between the number of resources on a page and the total amount of bytes needed to load page including its resources.

3.2. Zipf

One of the trends seen in web caching is that a few popular files are requested by a large number of users [19]. [7] initially confirmed that content requests follow a Zipf-like distribution by analyzing six traces from academic, corporate and ISP environments. Many different works such as [23, 1, 4, 9, 11, 34, 39, 36] have used this distribution in their own experimentation or analysis. Some of these works have noted how the Zipf distribution does not accurately reflect the popularity behaviour of the long tail [11, 8]. Content popularity on the tail seems to drop much faster than Zipf predicts. However most works still opt to use the Zipf-like distribution to allow for more direct comparisons with other works.

The Zipf distribution describes how the popularity of items, such as words in a language, or in our case number of requests for a specific resource, follow the distribution of

[6] wikipedia.org/wiki/List$_{\text{offileformats}}$.

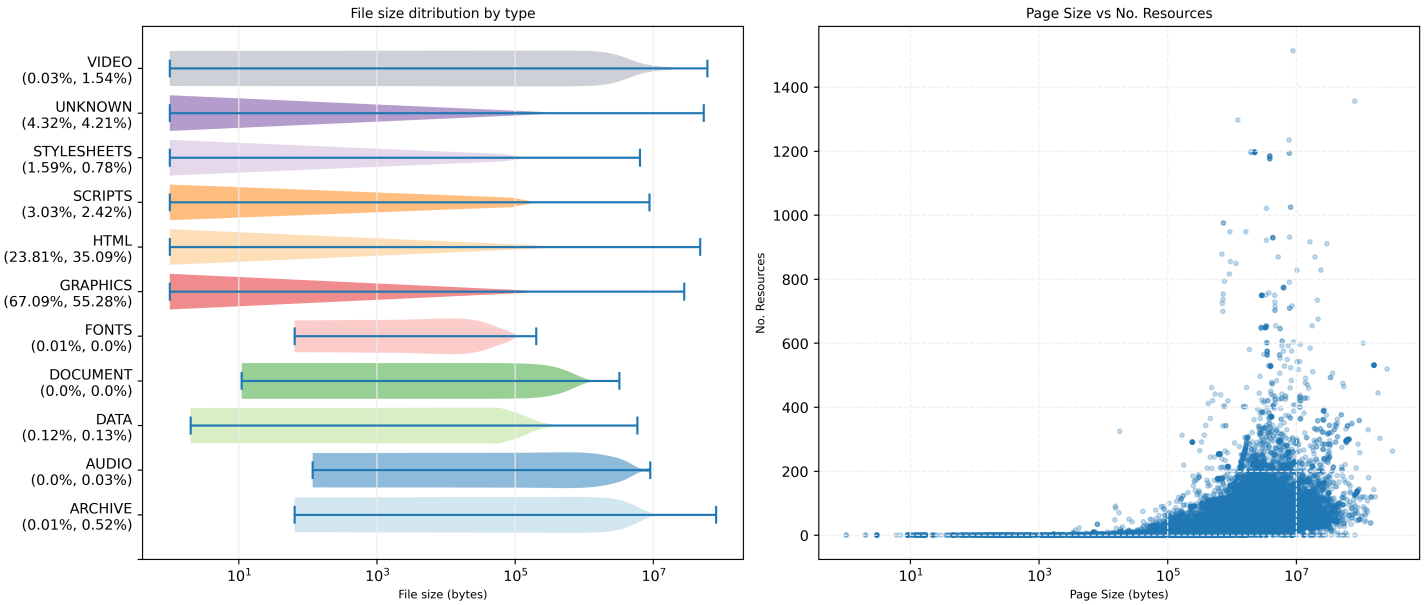


FIGURE 3.3: *(*Left*) Violin plot of the content size distributions for different content types. The labels on the y axis are annotated with the relative size of the type compared to the dataset in both no. of items and size in bytes. (*Right*) Scatterplot of required resources per page versus the size of those resources summed.*

$\frac{1}{n}$ where n is the rank of the item. An item of rank 2 is therefore half as popular as an item with rank 1 while an item of rank 3 is only $\frac{1}{3}$ as popular as the item with rank 1. The resulting effect is a limited set of very popular items and a long tail of less popular items.

We can adjust the rate of the popularity decline by changing a factor s that acts as an exponent to n, resulting in the following formula:

$$\frac{1}{n^s}$$

This adaptation is equal to the original Zipf distribution when using an exponent of $s = 1.0$ but becomes Zipf-like when using any other exponent. The effect of an exponent > 1.0 is an increase in the rate at which the denominator increases and thus a faster drop-off in popularity while an exponent < 1.0 results in the reverse effect.

The work of Breslau et al. [7] found that the content requests in the analysed datasets, follow a Zipf-like distribution with an exponent between $0.64 - 0.83$. [11, 9] chose to use an exponent of `0.8`, [26] uses an exponent of `0.9`, while [1] uses an exponent of `0.6`. Areas related to web requests such as torrents see a similar distribution in the popularity of torrents [27]. Erman et al. [12] analyse a billion HTTP requests over two days (in 2010) and find a Zipf-like distribution with an exponent around `0.88` for the content popularity. The work of Woo et al. [33] analyses a set of flow captures recorded over a week from a large cellular ISP in South Korea in 2013. They similarly find that the HTTP objects in the captures follow a Zipf-like popularity distribution with an exponent around `0.88`.

A different study with access to several base stations of a Turkish mobile network in 2015 showed a significantly different fit of the Zipf-like curve at `s=1.36` [5]. This fit seems to mostly match the body of the requests (after rank `100-200` out of the `16419`) where the head of the popularity graph shows a more gradual slope.

Given these more recent datapoints we have to consider whether the steepness of the Zipf-like distribution for web requests has changed since the original analysis in 1999 [7]. If we assume the steepness of the popularity drop-off has increased we could conclude that the most popular content is responsible for a larger portion of the total requests than it did in 1999. This shift could be related to factors such as the increased ease of content sharing that allows content to become extremely popular in short periods of time.

3.3. Trace Generation

Using the dataset collected in the previous section we can start to generate our request traces from the available pages and resources. In this section we discuss what a trace should look like and what kind of actions are possible.

3.3.1 Building Blocks

The main aspects of the trace, for our use case, is to model requests and user mobility. This is condensed in 3 different actions `ConnectUserTo`, `DisconnectUserFrom`, and `FetchResourceFor`. In addition to these actions we also model two actions to be able to control time and to request a capture of the current statistics actions: `SetIteration` and `TrackStatistics`. In the following sections we explain these actions in more detail. We also included a diagram to explain the relationship between the different actions in Figure 3.4.

1. Connect User To

   `CONNECT <user_id> <node_id>`
   While starting out or when moving around a user needs to set up a connection to the closest edge node. The `node_id` should be some general identifier for a node like `edge1` that can be mapped to a simulated node upon execution of the trace. A user can connect do different nodes by calling the `CONNECT` action multiple times.

2. Disconnect User From

   `DISCONNECT <user_id> <node_id>`
   Disconnect from a specific edge node, defined by `node_id`, that the user is currently connected to. If the `node_id` is the last connected node for this user the action should be followed by a `CONNECT` action before a new request can be sent.

3. Fetch Resource For

   `REQUEST <user_id> <node_id> <content_identifier>`
   Requests a specific piece of content by its identifier (such as a URL) from a specific

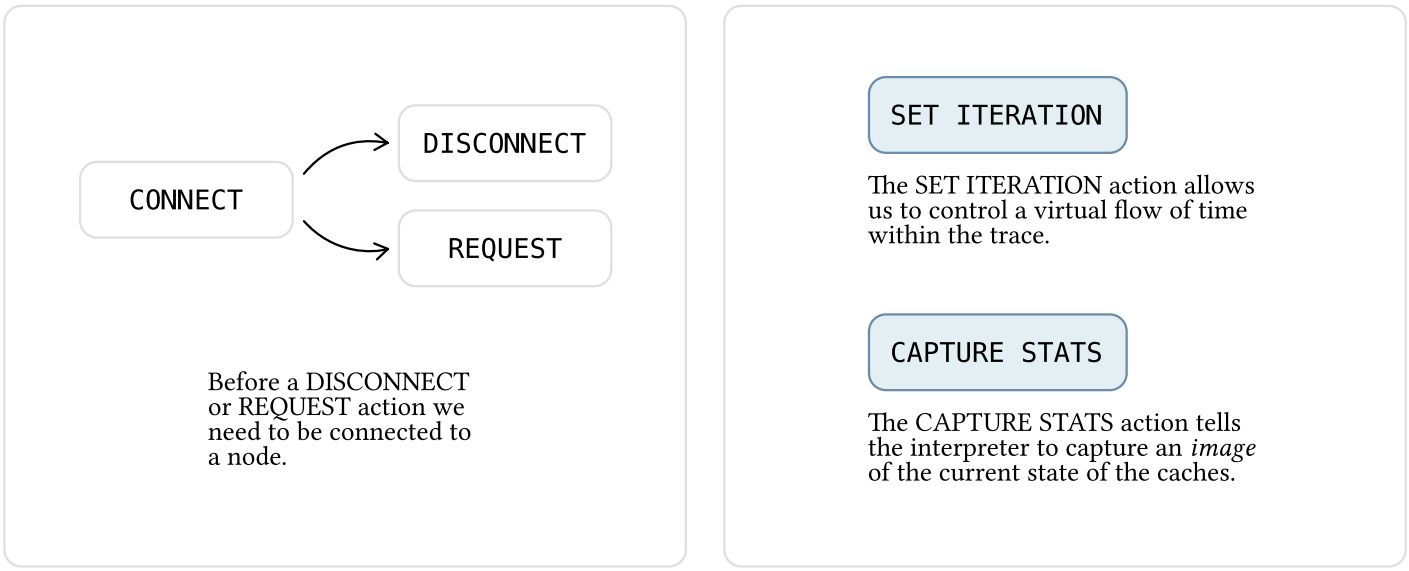


FIGURE 3.4: *The different building blocks used in request traces. On the left we show the primary actions a user can take, while on the right we show the actions used to control the simulation.*

`<node_id>` that the user is connected to. We explicitly mention the `node_id` again as a user is allowed to be connected to multiple edge nodes.

4. Set Iteration

   `ITERATION <iteration_no>`
   An iteration can be defined as a single loop where all users had a chance to request a resource. This variable allows us to abstract away from the underlying system time and splits up the simulation into sections that can be analysed separately.

5. Track Statistics

   `GET_STATS`
   Fetches the current statistics for each edge node and appends them to a file. This is usually executed after each iteration.

3.3.2 Writing a request trace

These instructions together make up a request trace that we need to evaluate a cache strategy. By writing the request trace to file we create a separation between our simulation- and evaluation step. It acts as an interface between the two. This allows us to swap out one of the two for an improved version in the future without the need to rewrite the other part. For example if we get our hands on a CDN based dataset in the future we could extract the requests to the same format used as the request trace and rerun our experiments. In addition to this the traces allow investigation of every single action executed by the evaluation for debugging, evaluations on the trace itself, or for the evaluation of Belady's MIN algorithm.

An extracted section of one of the request traces used is included in Figure 3.5. In this figure we show a short extract from the initialisation phase, where users connect to the edge node on their starting location, a set of requests, and finally the use of the `GET_STATS` and `ITERATION` instructions used to separate the trace into smaller chunks.

During initialisation, each user connects to their starting node

Users request all content required for the web page they visit in an iteration

After each iteration we record the statistics and increment the iteration

```
CON 8 edge1
CON 10 edge2
CON 13 edge1
<actions left out for readability>
REQ 8 edge1 static.iask.cn/v20210518/css/base5.min.css
REQ 8 edge1 static.iask.cn/v20210518/js/new/404.min.js
<actions left out for readability>
REQ 10 edge2 m3.auto.itc.cn/car/theme/auto_pv.js
REQ 10 edge2 db.auto.sohu.com/model_2266/photo/iframe.html
<actions left out for readability>
GET_STATS
ITERATION 3
```

FIGURE 3.5: *An annotated extract from an example trace generated*

3.4. Modelling User Behaviour

Now that we have defined what a trace file looks like we need to decide on a method to generate the actual requests and user movement. We achieve this by simulating a set of users that request content and move around a graph of edge nodes. The request behaviour is interchangeable, which allowed us to experiment with three different methods for the request generator. In section 3.6 we evaluate these different generators and select the configuration to use in our experiments.

3.4.1 Random

As a baseline we implement a generator that on each call selects a random resource to request. This generator is expected to have a mostly uniform popularity distribution which is not likely to be representative of real world traces.

3.4.2 Zipf-like

In this implementation we emulate the Zipf behaviour by creating a global page ranking of all the pages in the site-graph. Each resource is assigned a rank from `1...N` where `N` is the number of available resources. Weights are assigned to each resource by its rank and the exponent `s` in the Zipf-like function:

$$\frac{1}{n^s}$$

Every iteration a resource is selected using a weighted random selection based on these weights. This a commonly used strategy to emulate content popularity used by works such as [1, 23].

We can modify this generator by altering the **Zipf exponent** `s` to change the rate at which content popularity declines as explained in section 3.2.

3.4.3 Page Map

In this strategy we simulate a browsing experience for each user. When a user visits a page they will be able to, on the next iteration, follow any of the outgoing links for that page, as defined in the site-graph, or to move back in their browsing history. This is visualised in Figure 3.6. In addition we only provide each user with a subset of all the site-graph to emulate a specific interest for content. This means that for each user we keep track of the currently visited page and the subset of pages that are of interest.

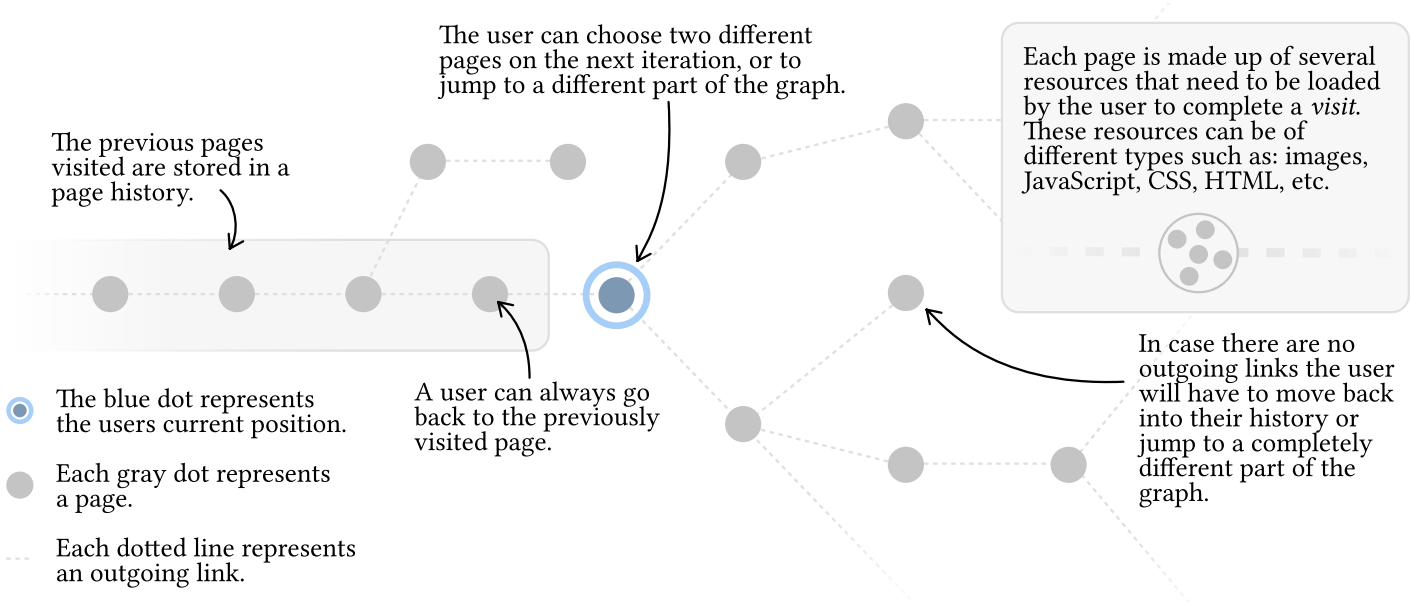


FIGURE 3.6: *A visual overview of the page-map from the perspective of a user.*

By implementing this behaviour, while simplified compared to reality, we try to emulate how a user would browse the web. The resulting content requests originate from the required resources for the each visited page.

In addition to following links we also allow our users to jump to a completely different page in their specialisation. This behaviour emulates search-engine like behaviour or external effects such as the sharing of a web page

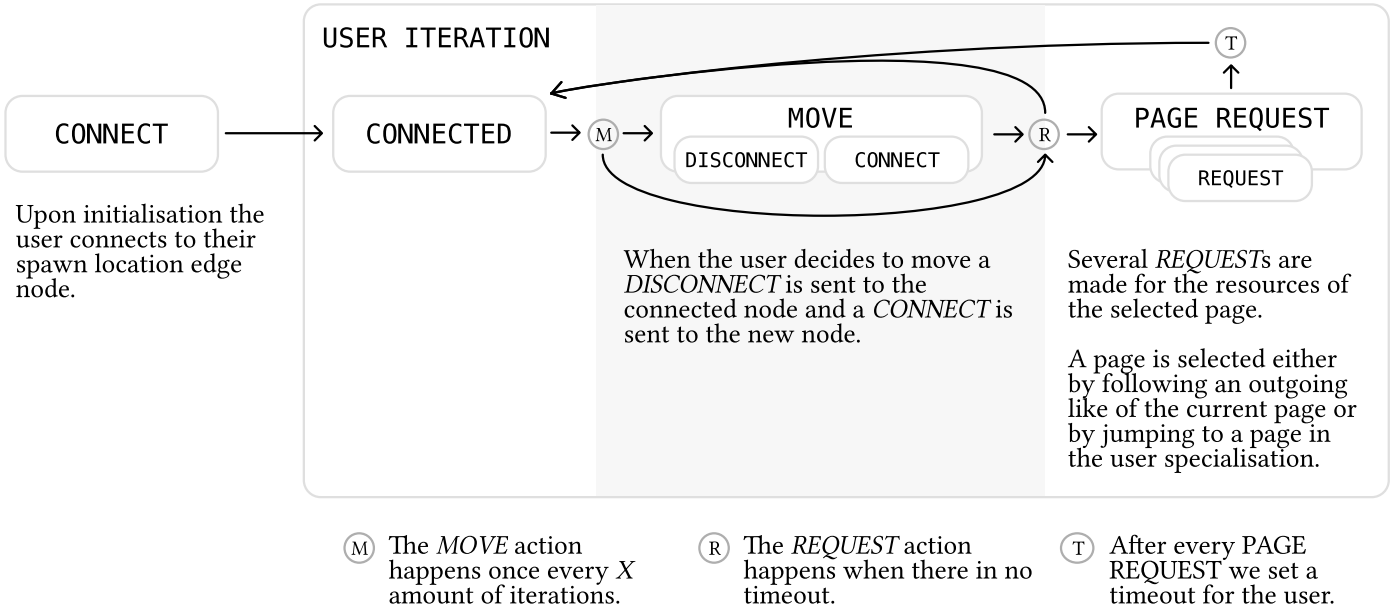


FIGURE 3.7: *A visual overview of the logic behind the page-map generator for a user iteration.*

A summary of the page-map behaviour is shown in Figure 3.7.

We can adjust this behaviour by altering the following characteristics:

Content Scope Each user is assigned a certain part of the site-graph that it can 'walk' around on. The smaller the scope the more specialised each user will be. A smaller scope will result in more predictable behaviour for each user in terms of pages visited (and resources requested). The scope is defined as a value between `0-1` where `1` includes the whole original set of pages while a value of `0.2` includes only a fifth of the dataset, chosen at random.

Jump Navigation This value, set between `0-1` defines the likelihood that a user will jump to a different piece of content in their specialisation. The value of `1` indicates that the user will always jump while a value of `0.2` means that on average every one in five page requests will be caused by a jump. A higher chance of jumping results in less predictable user behaviour and reduces the likelihood of similar requests due to shared dependencies.

Timeout Due to the large amount of resources used for every page visit we introduce a timeout that makes sure the resulting number of requests is similar to the other generators. The timeout parameter defines the highest timeout possible in iterations. After each page visit a timeout will be set for the requesting user based on a uniform-random distribution between `1-<timeout>`. A low value will result in more active users while a higher value will allow users to *sleep* for longer periods of time. We use a uniform distribution because it is easy to reason about. For example we set the default timeout to `12` based on an average of `6` resources per page to equal out the number of requests made with other strategies.

3.4.4 Movement

In this work we decided to represent user movement through a node graph. This graph is set up manually and represents the ways users can move between edge nodes. The different node maps used will be described in section 4.3.

Simulating realistic user movement is a complex task, "there is no tractable mathematical model available to describe user mobility, which impedes the understanding of user mobility in cache-enabled cellular networks" [19]. Therefore we opted to use a model based on a random uniform process. Once decided that the user will move a selection will be made at random from one of the neighbouring edge nodes on the node-graph.

We can adjust this behaviour by altering the **Mobility Speed**, a value between `0-1` that determines the likelihood of a user moving to the next node. A speed of `1` indicates that the user will move on every iteration while a speed of `0.2` means that the user will move on average once in every five iterations.

3.5. Configurable Aspects

Apart from the different configuration options for the generators there are some shared options that apply to each trace:

Number of Users Increasing or decreasing the number of users will change the amount of requests sent to the edge nodes, this is related to the generated traffic.

Content Map The content map defines the pages and their respective resources, altering this map alters the choice of resources for any generator.

Number of iterations The number of iterations determine how many times each user will be able to request content. Many different caching strategies will require some 'warm up' time to properly fill up the available cache space or generate a ranking according to which content is stored. Therefore the number of iterations should be chosen in line with the cache capacity for each node such that the strategy is able to fully warm up within the simulated number of iterations.

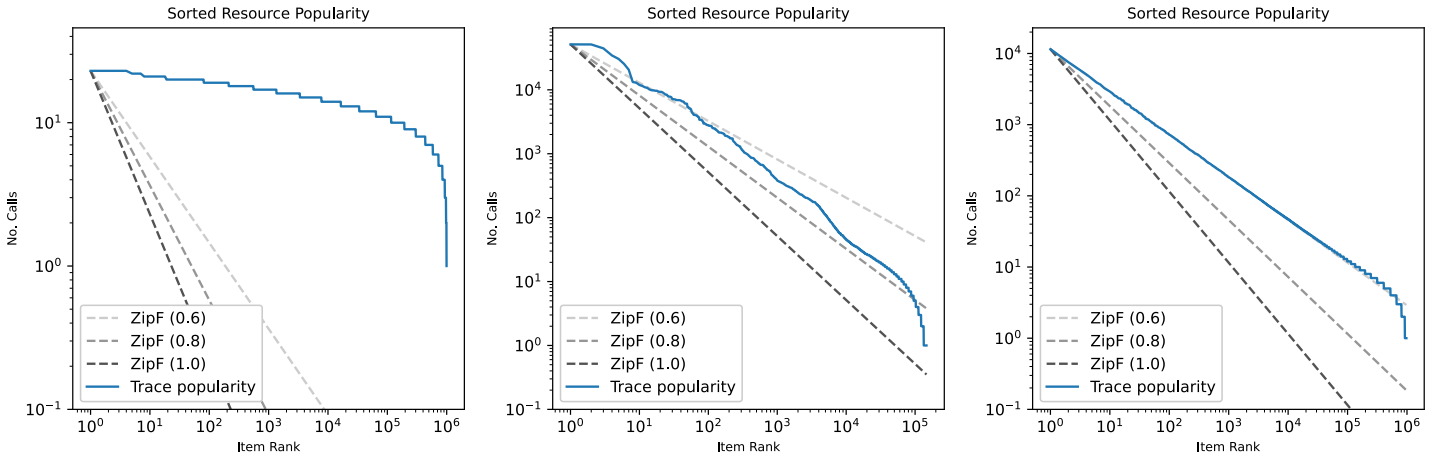
Seed Each trace should be reproducible by providing the same seed again, this makes our results reproducible for others and helps with debugging situations.

3.6. Evaluating Traces

For our evaluation of the generators we plot a the characteristics of a trace using a fixed seed. We then generated a set of plots that showcase different characteristics of these traces. These characteristics are used to determine which generator is suitable to use for the rest of this work.

3.6.1 Sorted Resource Popularity

First we look at the *sorted resource popularity* of each trace. By sorting the resource popularity we can provide each resource with a rank. This data of popularity versus rank is plotted on a log-log scale for the different trace generators in Figure 3.8. We make use of the *log-log* scale to be able to make a visual comparison with the Zipf-like distributions.



(A) *Random Generator* (B) *Page-Map Generator* (C) *Zipf Generator ($s = 0.6$)*

FIGURE 3.8: *Plots (a-c) show the cumulative distribution of requests for different trace generators*

In Figure 3.8a we can see how the resource popularity of the random generator does not match any of the chosen zipf-like distributions. Instead we find that the popularity drops very slowly. This is inline with expectations as resource popularity is based on a uniform distribution.

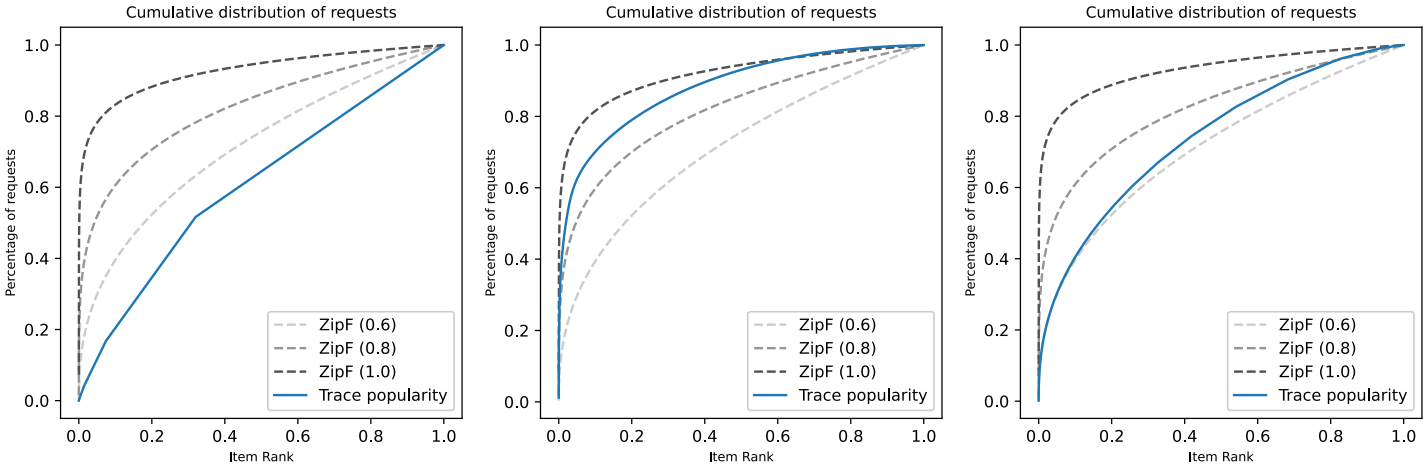
The Zipf-generator, seen in Figure 3.8c shows an almost perfect alignment with the Zipf-0.6 like distribution. This is expected as the requests are generated based on the Zipf-like distribution with `s = 0.6`.

Figure 3.8b shows how the page-map generator, browsing the web by following links, generates an interesting popularity graph. A group of highly popular resources (until `rank=80`) is followed by a Zipf-1.0 like drop-off in popularity, while the tail ($rank \geq 70.000$) drops off even faster. This drop-off on the tail is an interesting aspect that

cannot be simulated by the Zipf-like generator [11]. However, this attribute is found in more recent analysis on torrent popularity [27], and YouTube video popularity [8].

3.6.2 Cumulative Request Distribution

Figure 3.9 shows the cumulative request distribution to visualise how many resources are responsible for a given number of requests. In a similar fashion to Figure 3.8 we include the Zipf-like distributions for `s=0.6,0.8,1.0` for comparison.



(A) *Random Generator* (B) *Page-Map Generator* (C) *Zipf Generator ($s = 0.6$)*

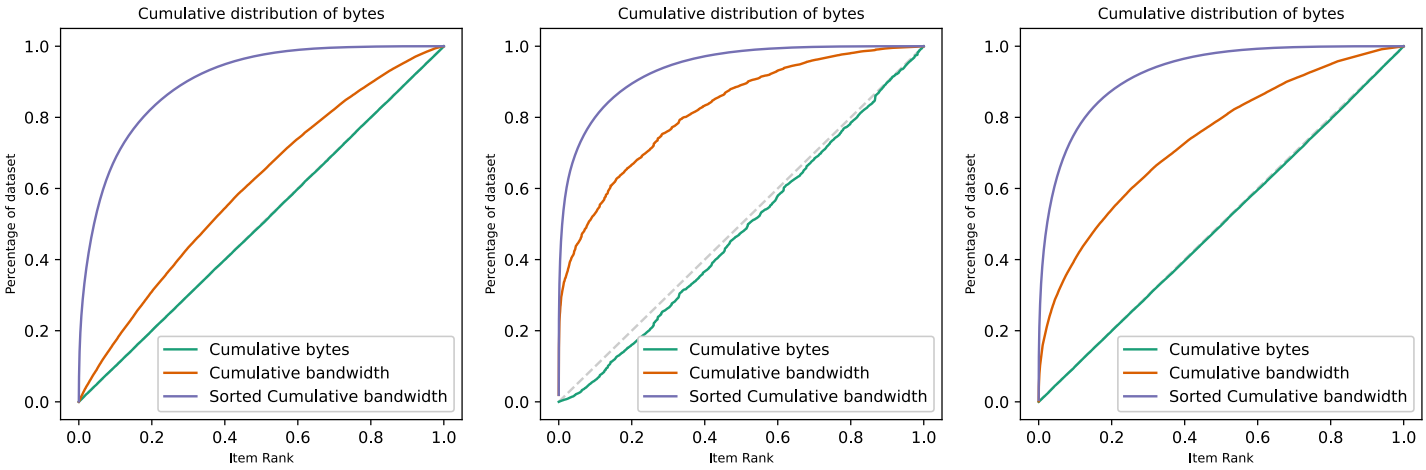
FIGURE 3.9: *Plots (a-c) show the sorted resource popularity for different trace generators*

These graphs show the relationship between the requests made by a trace compared to the popularity ranking of the resources in that trace. For example the random distribution, in Figure 3.9a, shows that we could serve 60% of the incoming requests by storing 40% of the most popular files. For our page-map generator, in Figure 3.9b, we can serve a similar 60% of the requests with only 5% of the most popular content. And finally in our Zipf-like generator, in Figure 3.9c, we can serve 60% of the request with about 30% of the most popular content.

3.6.3 Cumulative Byte Distribution

In Figure 3.10 we look at the relationship between the rank, the size of the content, and the amount of bandwidth generated. The sizes in bytes are normalised by the sum of bytes for all resources requested in the trace. The data is again visualised in a cumulative way to be able to reason about the values in terms of cache performance.

Each trace shows a mostly linear relationship between cumulative resource size and



(A) *Random Generator* (B) *Page-Map Generator* (C) *Zipf Generator ($s = 0.6$)*

FIGURE 3.10: *Plots (a-c) show the cumulative distribution of bytes for different trace generators*

fraction of the total bytes for all resources requested in that trace. This means that the resources are equally divided in terms of their byte sizes.

When multiplying this resource size with the associated popularity (number of calls) we can plot the total fraction of the bytes requested that are spent on a specific resource. Here we normalise our values by the total number of bytes requested during the trace. By plotting this value in a cumulative manner we show the relationship between the top ranking items (based on number of hits) and the number of bytes they represent in the total bandwidth of the trace.

We also plot a third line where instead, we base our ranking on the total number of bytes that are being served for a specific file. By sorting on the total number of bytes we get a better picture on which fraction of the resources causes the most traffic.

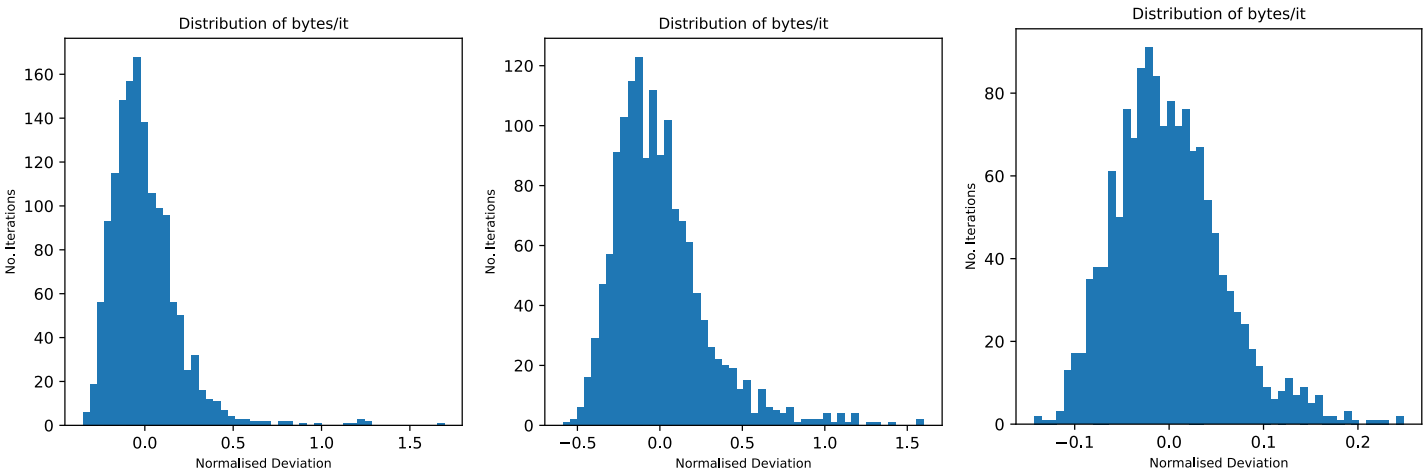
3.6.4 Bandwidth Deviation

In our final analysis we look at the uniformity of the bandwidth over the simulated iterations for each generation. We do this by visualising the deviation of the average bandwidth in a histogram for each generator. The deviation is calculated by subtracting the average bandwidth per iteration and normalising the result by this average.

For all generators the deviation seems to be skewed to the positive (more bytes) with some outliers that request almost double the average number of bytes. The Zipf distribution is more condensed but also shows some skew towards positive outliers.

3.6.5 Conclusion

Based on these characteristics we can make a decision on the type of generator to use for the rest of this work. The random generator is not considered due to the mostly uniform



(A) *Random Generator* (B) *Page-Map Generator* (C) *Zipf Generator* ($s = 0.6$)

FIGURE 3.11: *Plots (a-c) show the cumulative distribution of bytes for different trace generators*

popularity generated, this does not match the expected popularity curve as found by previous work.

The *user* and *Zipf* generators both generate resource popularity that matches the Zipf-like distributions found in real world traces in [7]. The *user* simulation also contains attributes like a steeper tail drop-off seen in more recent works [27, 8].

Because of the absence of representative traces we cannot with certainty claim that one of these generators reflects the real world more accurately than another. Therefore we decide to include three different configurations in our experiments:

- Based on [7] we will include a Zipf-like generator with an exponent of `0.75`.
- Based on certain similarities between our page map generator and the works of [27, 8] we include the page map generator with a content scope of `0.1`, jump navigation set to `0.05`, and the max timeout set to `12`.
- Based on the work of [5] we include a Zipf-like generator with an exponent of `1.3`.

With our evaluation we now answer **RQ1**: "*How can we create a simulated environment for cache performance testing?*". To track the performance metrics we are interested in, *Hit-Ratio* and *Backhaul Savings*, we need to evaluate our cache between a set of users, that send requests, and an origin server that hosts the resources. However due to the absence of representative traces we need to model our own users and request generators that match previous work. We evaluated three options for trace generators and compared them to characteristics of traces found in previous work to select a total of three configurations that we will use for the experiments in this work.

3.7. Experimental Setup

As we focus on the hit-ratio and backhaul savings in our caching strategies we do not need a network setup to measure statistics like latency or throughput. This simplification allows us to create a simulated environment where actions are passed as function calls to the virtual implementation of the caching strategies.

The implementation for our simulation, generators, and any of the performed experiments are available, open source, at github.com/nbelzer/msc-thesis. All experiments in this work are described in this repository in the form of Jupyter Notebooks with matching standalone Python scripts that were used to run the large scale versions. All experiments were evaluated using Python version `3.9.5`, any package dependencies are pinned in the included `requirements.txt` file.

To be able to simulate a large amount of users and edge nodes we utilized the DAS5 Cluster [3]. Specifically we used the nodes in the TUD (Delft University of Technology) cluster, which consists of dual 8-core CPUs (primarily Intel E5-2630v3) running at 2.4GHz and 64GB of memory[7].

The DAS5 Cluster was used to collect our dataset, and perform any of the simulations that are presented in the following chapters.

[7] `https://www.cs.vu.nl/das5/clusters.shtml`

SECTION 4

The effects of a limited population

In this section we evaluate the effects of a limited population on the performance of a cache by making use of our simulation and the generated traces. In this section we aim to verify the findings of previous work like [11, 17, 21] where a reduced population per node reduces the performance of a cache.

To answer this question we will look at a traditional reactive cache strategy using the *least-recently-used* eviction strategy. In addition to this strategy we will also use Belady's MIN algorithm [6] to evaluate the effects of a more constrained scope on content popularity.

4.1. The Baseline Strategy

The baseline strategy that we will use for our experiment is a traditional pull-based cache. This means that content is only requested from origin once a user makes a request for that content. The cache uses the *least-recently-used* eviction strategy to make space for new content. This, however, allows a single request for a large file to potentially evict a large fraction of the stored items, causing a lot of cache misses [11]. Therefore a *cache-threshold-policy* is used to limit what files are cached. The policy used in our baseline will only cache a piece of content after `3` requests.

4.2. Belady's MIN Algorithm

Belady's MIN strategy allows us to calculate the theoretical optimum for a specific trace file. The algorithm minimises the amount of fetches from the origin by using an eviction policy that evicts the item that will not be used for the longest period of time [6]. Doing this, however, requires *a priori* knowledge of the requests (which makes it unsuitable for a real world situation).

To be able to compare the output of the MIN algorithm with our own strategies, we extended Belady's MIN algorithm with an understanding of item weight as not every item takes up the same amount of storage in our cache. A maximum weight is set that is used to determine when to evict items. The size, in bytes, of an item is used to determine the weight. The maximum weight of the cache is set to the cache capacity (in bytes).

By allowing different weights for items stored in the cache we can no longer infer that the result of the algorithm is the theoretical optimum. It could be more optimal to choose not to store an item with a large weight if that leaves space for lower weight items that are retrieved more often

Table 4.1a shows the optimal caching strategy for the files `A`, `B`, `C`, and `D` with weights

Table 4.1: *Caching strategy examples*

(a) *Optimal strategy with weights*

	A	B	C	D	C	A	B	C	D
@1	A	-	C	-	·	(A)	-	·	-
@2	A	-		D	-	(A)	-	-	·
@3		B	-	-	-	-	·	-	-

(b) *Belady's MIN with weights*

	A	B	C	D	C	A	B	C	D
@1	A	-	-	D	-	A	B	-	-
@2	A	-	-			A			D
@3		B	C	-	·	-	-	·	-

`2`, `1`, `1`, and `1`, respectively. The optimal strategy results in 5 misses (or fetches from origin) and 4 hits. In terms of weights we retrieved `7` from origin and served 4 from the cache.

In Table 4.1b we see the strategy that is chosen by our extension of Belady's MIN algorithm. The main difference is that the optimal strategy does not try to keep A in cache. The strategy results in 7 misses and 2 hits. In terms of weights we retrieve 9 points from origin and serve only 2 from the cache.

This inefficiency is a result of a restriction in Belady's MIN algorithm where content, upon a request, needs to be stored in a slot. This stems from the context in which Belady's MIN algorithm was designed, where content needs to be inserted into registers before it can be used. In our use case this restriction does not apply as we can choose to forward the response from origin to the requestee without saving the content in the cache itself. By lifting this restriction we avoid the eviction of content that is used before the item to be inserted is used again. For example when inserting `X`, we will not evict items that are used before `X` itself is used again. This means that if `X` is never used again it will not be allowed to evict any items. While if `X` is requested again immediately after this request it is allowed to evict any item.

Table 4.2: *Belady's MIN (with pass-through)*

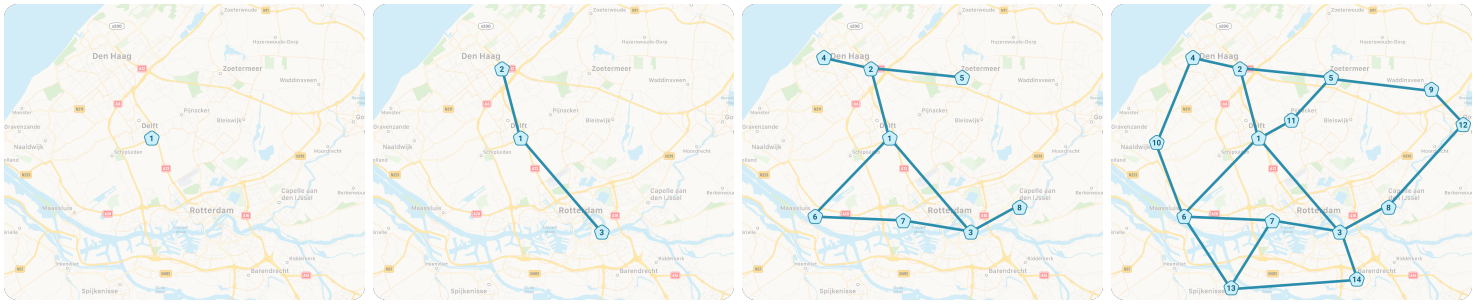
	A	B	C	D	C	A	B	C	D
@1	A	-	-	(D)	-	·			D
@2	A	-	-	-	-	·	B	-	-
@3		B	C	-	·	-	-	·	-

In the example shown in Table 4.1b this means that `D` is not allowed to evict `A` or `C` as these two are used before `D` is used again, but it would have been allowed to evict `B` if it had been stored by the cache. This behaviour also requires an additional check, before eviction, to verify that the items to evict reclaim enough weight for item `X` to be inserted.

Table 4.2 shows this behaviour implemented which for our example results in 6 misses and 3 hits. While in terms of weights we fetch 7 points from origin and serve 4 from the cache. While not the same as the optimal strategy we will use this adaptation of Belady's

for our next experiments.

4.3. The Population Experiment



(A) *Map with 1 Node* (B) *Map with 3 Nodes* (C) *Map with 8 Nodes* (D) *Map with 14 Nodes*

FIGURE 4.1: *Images a-d describe the different edge setups used inspired by the different cities around Delft and the density of 5G antennas in these areas.*

Figure 4.1 shows four edge networks with a different density. While the locations of the edge nodes are fictional we based the locations on the cities and densities of 5G antennas[8] in the area. The lines between the nodes show the internal links between the edge nodes. We assume that every edge node has its own backhaul link.

To determine the effect of the limited population we perform the same experiment for each of these setups. We will evaluate both the baseline strategy, LRU, as well as Belady's MIN algorithm. For each setup we keep the number of users and iterations the same. Larger setups (as shown in Figure 4.1) will therefore have a lower population of users per node which should cause the decrease in performance as described by [11, 17].

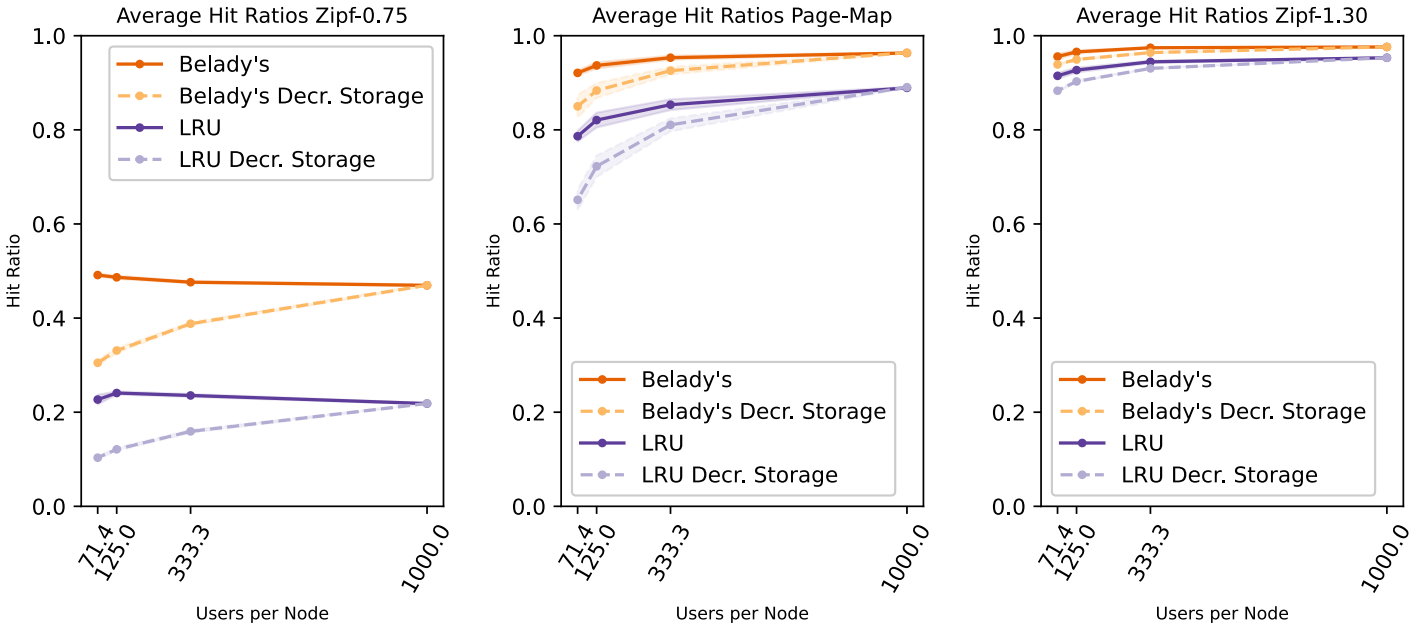
We introduce another variable by changing the storage capacity that splits our experiment into two scenarios. In the first scenario we keep the storage capacity per node the same as the number of nodes increases. Larger setups therefore simulate an expanded edge network with more storage in total. In the second scenario we decrease the capacity of the nodes with the number of edge nodes. This simulates the choice between multiple smaller nodes compared to a single large node for a given area as the total storage capacity remains the same.

These different setups are evaluated using traces generated by the three different generators outlined in section 3.6.5. Each trace contains 1000 users making requests over 5000 iterations. In the first scenario we keep the storage size for each node the same at `1024MiB` while in the second scenario we distribute the total storage capacity of `1024MiB` uniformly over the edge nodes. We evaluate each trace on all three setups and repeat the experiment 10 times (each with a trace based on a different seed) to improve the

[8]Retrieved from antennekaart.nl/kaart/5g.

confidence in our results.

4.3.1 Results



(A) *Zipf-0.75 generator* (B) *Page-Map generator* (C) *Zipf-1.30 generator*

FIGURE 4.2: *Limited Scope Hit Ratio results for the different generators. Each plot shows the average number of users per node versus the average hit ratio achieved.*

Figure 4.2 shows the results of our simulation for the three different generators. The x-axis on each plot shows the average number of users per node, this is based on the amount of users in the trace divided by the size of the node setup. This means that the ticks on the x-axis match the edge node setups in Figure 4.1 in reverse order. The y-axis shows the average hit-ratio for the edge nodes. The error band is used to visualise the standard deviation from the average hit ratio. For both Belady's MIN and LRU we plot two scenarios, one where the storage size per node stays the same and one where the storage size per node decreases as we increase the number of edge nodes.

Among the three generators, we see that the strategies for the trace generators Page-Map and Zipf-1.30 show a decrease in performance when the average number of users per node decreases.

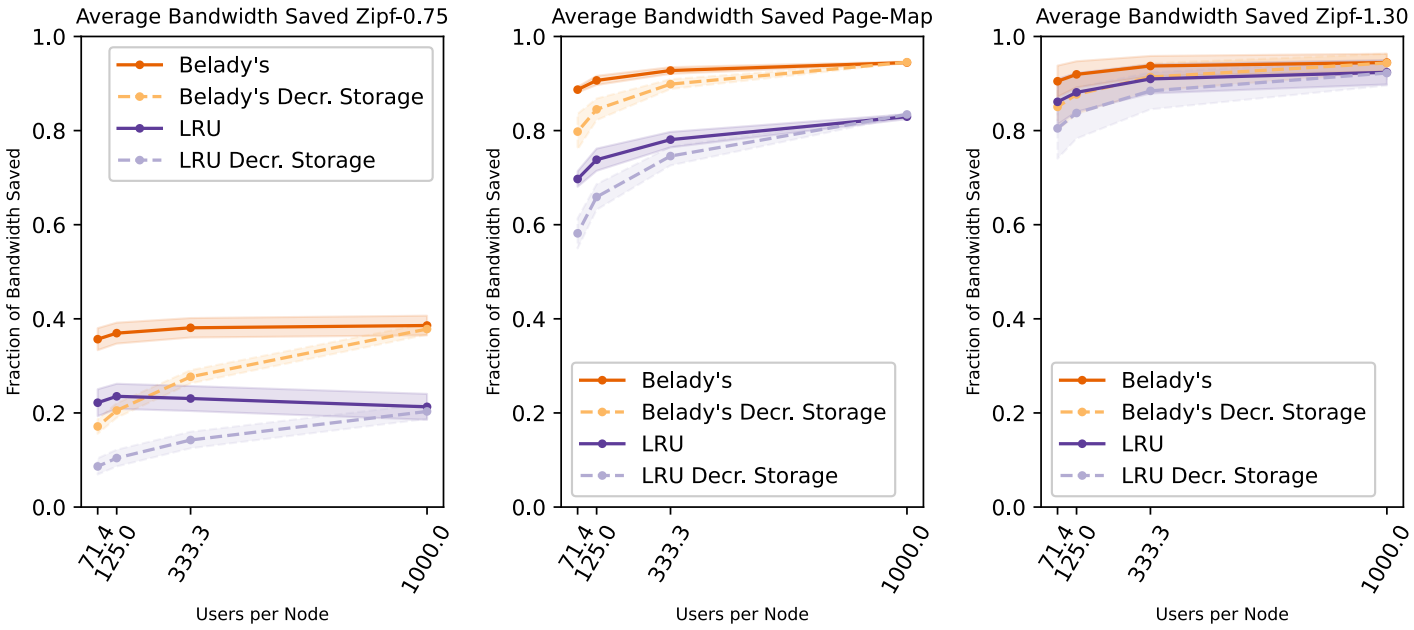
For the Zipf-0.75 generator we initially see no decline in performance. As seen in section 3.6 the lower exponent used for this Zipf-like distribution results in a more gradual decrease in popularity. This means a larger portion of the resources are responsible for the majority of requests compared to the Page-Map or Zipf-1.30 generators. Together

with the fact that the total storage capacity of the setup increases this actually results in a performance improvement for lower population counts per node as more of the popular files can be stored.

When we decrease the storage size per node to match a total capacity of `1024 MiB` we do see a performance drop-off for the Zipf-0.75 generator and an increased drop-off for the Page-Map and Zipf-1.30 generators.

Interestingly we note that the LRU strategy is better able to keep up with the optimal performance outlined by Belady's MIN algorithm for traces generated with a higher Zipf-exponent as the gap between the two, in terms of performance, is smaller. It shows the impact of a steeper popularity drop-off. The smaller the fraction of the resources making up the majority of the requests, the better our cache performance and the less negative impact the population count has.

Next to the hit-ratio we also analysed the backhaul bandwidth savings (or bandwidth served by cache) achieved for the different generator strategies to see if there would be a similar correlation. The results are presented in Figure 4.3.



(A) *Zipf-0.75 generator* (B) *Page-Map generator* (C) *Zipf-1.30 generator*

FIGURE 4.3: *Limited Scope Bandwidth savings results for the different generators. Each plot shows the average number of users per node versus the average bandwidth savings achieved.*

We find a similar correlation for the bandwidth savings as for the hit ratios. The difference between the two shows in the errorband or standard deviation from the average result, for all setups the standard deviation increased. This effect stems from the fact

that each trace assigns a different priority to the resources from the dataset and the fact that the resources do not have a uniform size distribution.

Based on these results we can now answer **RQ2**: "*What is the effect of a limited scope on content popularity for a local cache node?*". As visible in Figure 4.2 and Figure 4.3 we see a negative performance impact on the cache performance when the ratio of users-to-nodes decreases, confirming the findings by [11, 17]. This effect is present, but less strong, when the storage to user ratio increases. We conclude that new proposals are required to limit the impact of this effects for edge-CDN deployments.

SECTION 5

Mobility Strategies

In this chapter we investigate how mobility data, for example, by listening for handover events in the mobile network, can be used to improve upon existing caching strategies.

We start out with our baseline strategy, which is similar to the baseline used in section 4. We expand upon this strategy by introducing cooperation between neighbouring nodes in the *Cooperative LRU* strategy. We then introduce the *Profiles* strategy that aims to reduce the performance drop-off caused by a limited population per node by moving popularity data with the users. In addition we describe a *Federated* strategy where multiple nodes work together by dividing the workload. Finally we introduce a *Hybrid approach* between the two mobility and Federated strategy.

5.1. Baseline Pull-Based LRU

Similarly to the baseline described in section 4.1, our baseline strategy represents a commonly used reactive, or pull-based, setup using the *least-recently used* eviction strategy [19]. In addition a *cache-threshold-policy* is used to limit which files are stored in the cache. We will use a threshold of `3` requests, where each resource will only be stored after the third request for it. Internally this requires the cache to keep a counter for each unique resource identifier. The threshold policy is designed to prevent large but unpopular content from evicting a lot of items that might have a higher global popularity [11].

When the cache has no available space for a new resource it will clear out the least recently used items until enough space is available. These items are thus evicted from the cache until they are requested again and pass the *cache-threshold-policy* again.

5.2. Cooperative LRU

The Cooperative LRU strategy enhances the baseline strategy by altering the implementation. Instead of requesting the content from origin upon a cache miss we cooperate with the neighbouring edge nodes [26]. In our strategy we make use of the mobility data in the mobile network to narrow down cooperation to nodes a user has recently visited, their *node-trail*. All successful requests will decrease the amount of requests to origin and reduce traffic on the backhaul at the cost of internal bandwidth between the nodes. A parameter `N` is used to set the *node-trail length*. This value limits the amount of previously visited nodes that will be considered.

The main assumption behind this strategy is that popular resources will be able to follow a user around as, due to the popularity, there has likely been a request for at one of the previous `N` nodes. If the resource is found on one of these nodes it is forwarded to

the requestee and stored locally for follow up requests or for the next cache the user will visit.

The same eviction strategy applies as in the baseline LRU. Similarly resources also need to pass the *cache-threshold-policy* before they are stored.

5.3. User Profiles

The work of [17] mentions how for smaller user populations, caches will only have a partial view of the global content popularity. This is considered the main cause for the reduced performance as seen in section 4. The Profiles strategy aims to reduce this effect by moving popularity data with the users in the form of a profile. Each user builds up a profile that keeps track of the latest `N` amount of requests that provide an edge node with a reasonable assumption on what content its users are most likely to request. Profiles moves with the users and are provided to each edge node they connect to.

Based on all connected profiles the edge node combines the requests into popularity data, or a ranking, based on the number of occurrences in different profiles. As the aggregation process is too costly to run in-flight with the incoming requests, connection- or disconnect-events it is executed every `5` iterations instead.

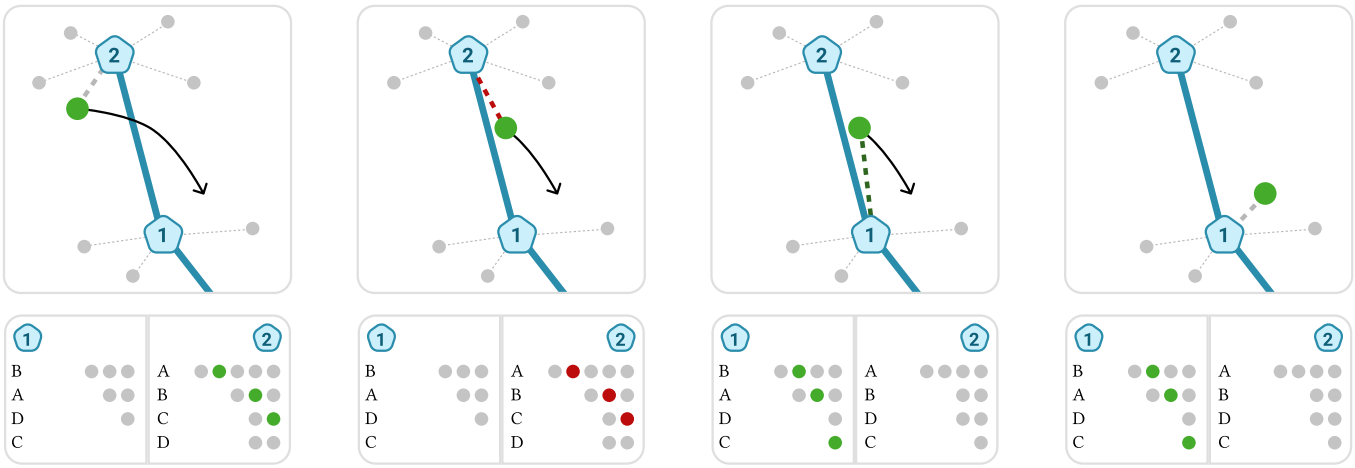
The ranking determines what resources are allowed to be stored locally. When the cache is full, new items are only stored if there are enough items with a lower rank to be evicted. If the item is not in the ranking it is given the lowest rank which, unless a cache has unused capacity available, results in the item not able to evict anything and not being stored.

In addition to the ranking, the aggregation process also creates a set of last known nodes for each item in the ranking. When a profile `A` contributes item `X` to the ranking we store the last known node for profile `A` as a potential location for `X`. When a request for `X` comes in, and the file cannot be served locally, we forward the request to the last known nodes for `X`. Upon a successful response we save the file on the local cache. We have visualised the behaviour of the Profile strategy during a handover in Figure 5.1.

To allow profiles to move with users, we either make use of an entity in the mobile network, or allow user devices to store the profile on device. Moving profiles between neighbouring edge nodes during movement is potentially limited when the user disconnects for longer periods of time (for example on planes). Depending on the size and scope of the profile it seems preferable to have the profile stored on device, providing the user with control over which data is shared and the ability to wipe the profile. Additionally, if a standard format for a profile is used we would avoid the need of different mobile networks to communicate with each other.

5.4. Federated

The federated strategy takes a different approach to content storage, instead grouping neighbouring nodes together to make better use of the available storage. This strategy



In this example scenario a user is connected to edge node 2. The user will follow the path outlined by the line that will require a new connection to edge node 1.

Below the map we show the live aggregated popularity of resources A, B, C, and D. As we can see the profile of contains items A, B, and C.

At this stage the user has started moving and is now in reach of 1. This allows for the start of a handover. The first step of which being a disconnect from edge node 2.

Upon disconnect the profle from is removed from 2. The effect of which is shown in the popularity tables as items A, B, and C lose 1 interested user.

At the same point the user will connect to 1. Upon connection the node will fetch the profile for and add it to the list of connected profiles. In addition we store that this user and profile came from edge node 2.

Once the path is complete the user is now fully connected to edge node 1. Once the next aggregation process on edge node 1 completes the ranking will include the resources for .

Once a request for a previously unknown resource arrives we look at the profiles that contain this resource and try to fetch the content from the nodes that previously stored this profile. In this example, for resource C, we would ask edge node 2.

FIGURE 5.1: *A visual overview on what happens in the Profiles strategy during a user handover. In each step we show the map with users and edge nodes, the current live aggregated ranking for each node, and an explanation on what is happening.*

eliminates any duplicate storage of popular content by splitting the workload between all available nodes based on a hash of the requested identifier. For this to work we assume that neighbouring nodes have high speed connections between them where we incur minimal latency. This makes the federated approach suitable for nodes that are physically close together.

5.4.1 Optimizing storage use

To make optimal use of the storage at each node within the group of nodes, files should be equally distributed. A hash function will uniformly distribute generated hash values across the output range. By splitting up the output range into N equally sized sections we load-balance any incoming request to a specific node. Due to the uniform nature of a hash function this results in a uniform distribution of the different resources over the nodes. By using the same hash function on all nodes in the Federated group there is no need for a centralised load balancer.

This works as long as the network of edge nodes is stable, no edge nodes fail or disconnect and no edge nodes join the group. A distributed hash table like Chord [28] is likely to be a better approach when the size of the network is considered variable, for example due to machine failures, to limit the effect of shifting the keyspace to the neighbouring nodes. While this would be crucial for an actual implementation in the real world, these details are left out as we are only interested in the potential performance that comes with such a setup. In our implementation when a node is considered inaccessible the requests are forwarded to origin. Origin could however be a different (more centralised) caching layer in the mobile network to prevent a large amount of requests hitting the actual content origin.

Python 3, by default[9], uses a `hash` function named SipHash2-4 [2] that maps to a keyspace of a signed 64 bit integer (on 64-bit builds). SipHash is designed to work with a secret key to prevent collision attacks. In Python this behaviour is encapsulated in the `PYTHONHASHSEED` environment variable that is set at random for every Python thread. For us this means that each thread needs to use the same `PYTHONHASHSEED` to generate reproducible results, or disable the feature by setting it to 0[10].

To verify the behaviour of this hash function we performed our own testing. Here we hashed a subset of identifiers from the retrieved dataset starting with 1/8 of the resources, picked at random, up to the full size of the dataset itself. The hashed values are then plotted in a histogram with 14 bins, representing the largest edge network we will use in our experiments. In this plot the y-axis shows the percentage of items in each bin. The resulting plot is shown in Figure 5.2 and shows that even for smaller subsets the distribution of files remains mostly uniform. We also confirm that there are no hash collisions for the identifiers in our dataset.

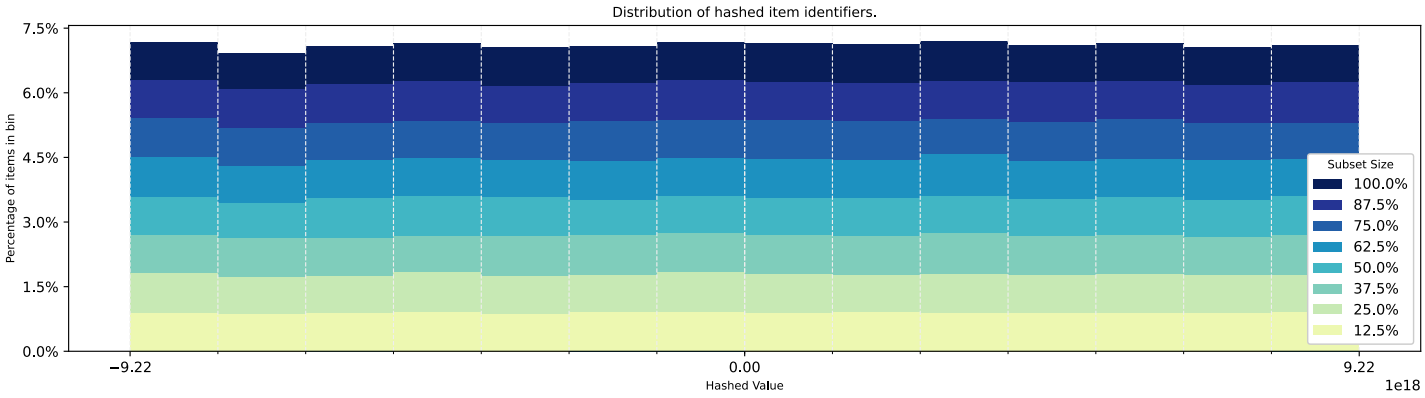


FIGURE 5.2: *Distribution of hashed item identifiers for different subset sizes of the dataset across 14 bins.*

[9] `https://www.python.org/dev/peps/pep-0456/`
[10] `https://docs.python.org/3/using/cmdline.html#envvar-PYTHONHASHSEED`

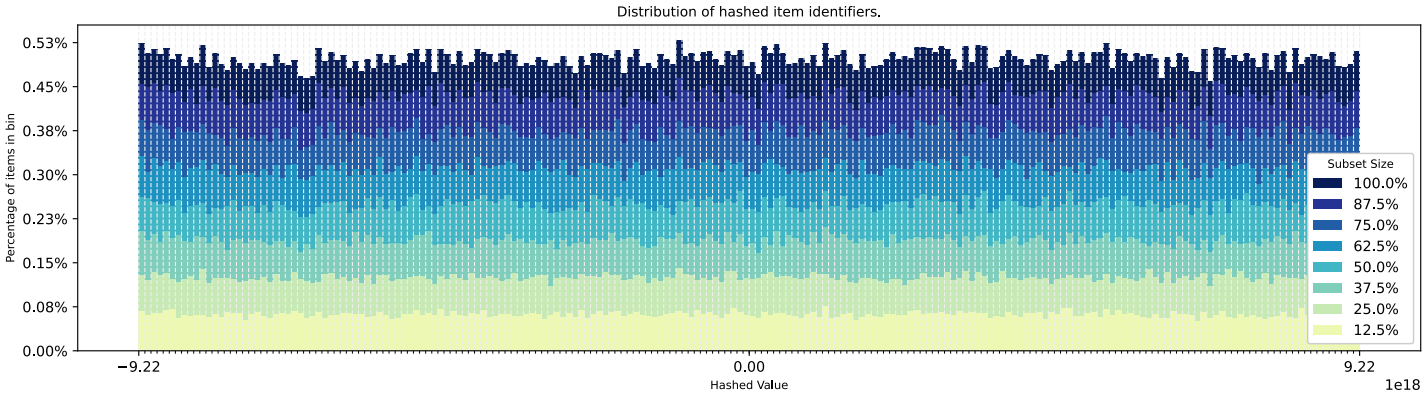


FIGURE 5.3: *Distribution of hashed item identifiers for different subset sizes of the dataset across 200 bins.*

Only when we increase the number bins do we start to see some deviation from the uniform distribution. This is shown in Figure 5.3 where we split the resulting values in 200 bins. However as the maximum edge size in our experiments is 14 nodes we not considered this an issue.

5.4.2 Content ranking and eviction

To be able to compare the Federated strategy with the other strategies we again make use of the LRU strategy for content eviction. The advantage over the previous strategies is created by the uniform distribution of content over all connected nodes. As each node is responsible for a subsection of the incoming requests the strategy is able to keep a lot more items in storage. This comes at the cost of tracking the least recently used item per subsection (instead of globally).

Even though the hashing algorithm provides a uniform distribution of the incoming requests there is no guarantee that the file sizes for these requests are also uniformly distributed. This likely results in a skewed distribution in terms of assigned bytes and therefore the amount of items that can be stored at each cache.

In our testing we find that there is indeed some variation between the number of bytes assigned to each bin. Similarly to our previous hash experiment, in section 5.4.1, we select differently sized subsets of the resources in our dataset. We then hash the resources in each subset to a selected number of bins. For each bin we look up the byte sizes for the assigned item to calculate the total bytes assigned. This is visualised in Figure 5.4 and Figure 5.5 where we use 14 and 200 bins respectively. While in the 14 bin scenario the deviation is minor, the 200 bin scenario shows significant outliers, even for smaller subsets.

This means there is the potential for an unequal distribution in terms of the latest timestamp for which content is still stored. One cache with a large number of small files might be able to store all files up to 50 iterations ago while a cache with a set of larger

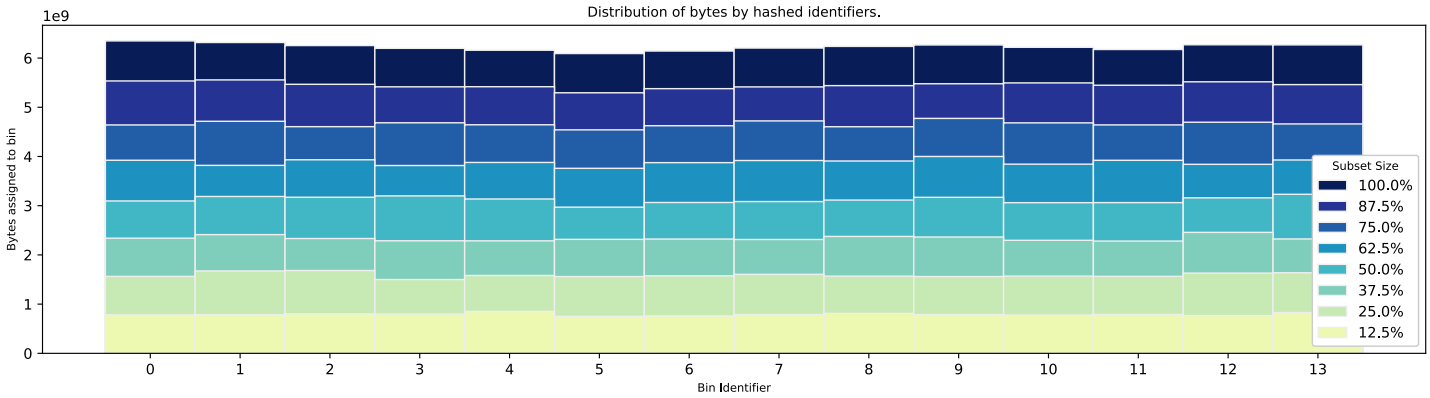


FIGURE 5.4: *Distribution of the bytes belonging to the hashed item identifiers for different subset sizes of the dataset across 14 bins.*

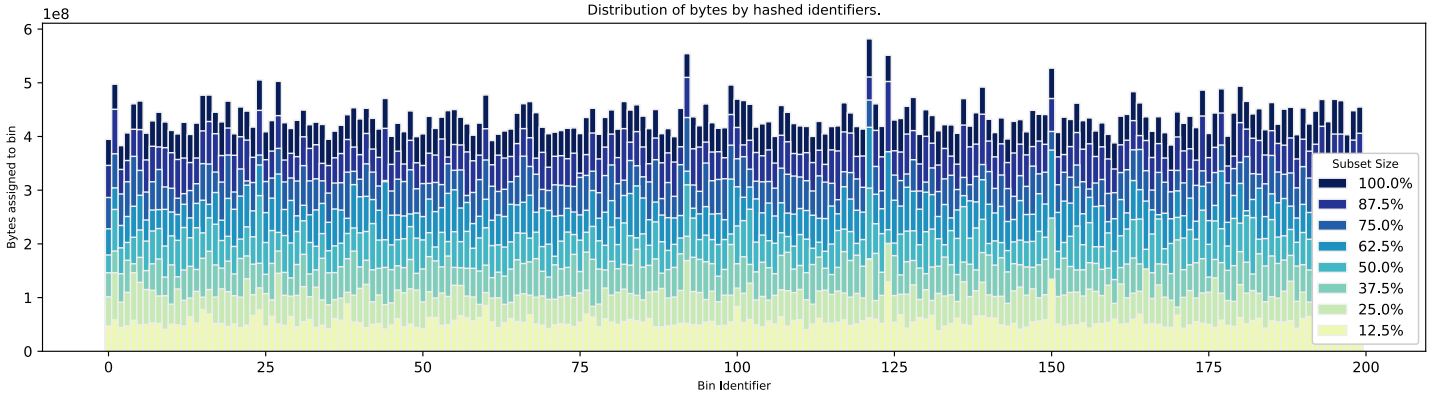


FIGURE 5.5: *Distribution of the bytes belonging to the hashed item identifiers for different subset sizes of the dataset across 200 bins.*

files might only be able to store files accessed up to 40 iterations ago. File A hashed to the first node will be able to stay in the cache for longer compared to File B hashed to the latter node.

This results in reduced performance for certain sections of the hash keyspace while the other sections will have improved performance as they are able to store more files in comparison.

As this behaviour is undesirable we could introduce a layer of abstraction representing an additional lookup. Instead of storing the files itself on the responsible node, determined by the hashing algorithm, the node stores the location of the file which could be a different node. This would allow total freedom in terms of content placement and a potential algorithm to better distribute content. For example storing new content on the node with the lowest number of files or the least recently used timestamp. This functionality however introduces additional complexity in the form of communication overhead. As the deviation is minimal in our largest setup of 14 nodes (see Figure 5.4), and because we would introduce addition complexity in our strategies, this functionality is not implemented in our simulation.

5.5. Hybrid

By grouping all the nodes together and forwarding the incoming requests to the appropriate node the location of the user (within the node group) is no longer relevant in the Federated strategy. However, as the Federated strategy relies on reliable and fast connections with the other nodes it should only be used with nodes in close proximity. At a larger scale the strategy will need to group edge nodes together into multiple virtual nodes or federated groups. At this point we are able to utilise one of the other two proposed strategies to handle the mobility of users between these virtual nodes. This type of hybrid approach will allow us to combine the best of the two types of strategies.

5.5.1 Substitution

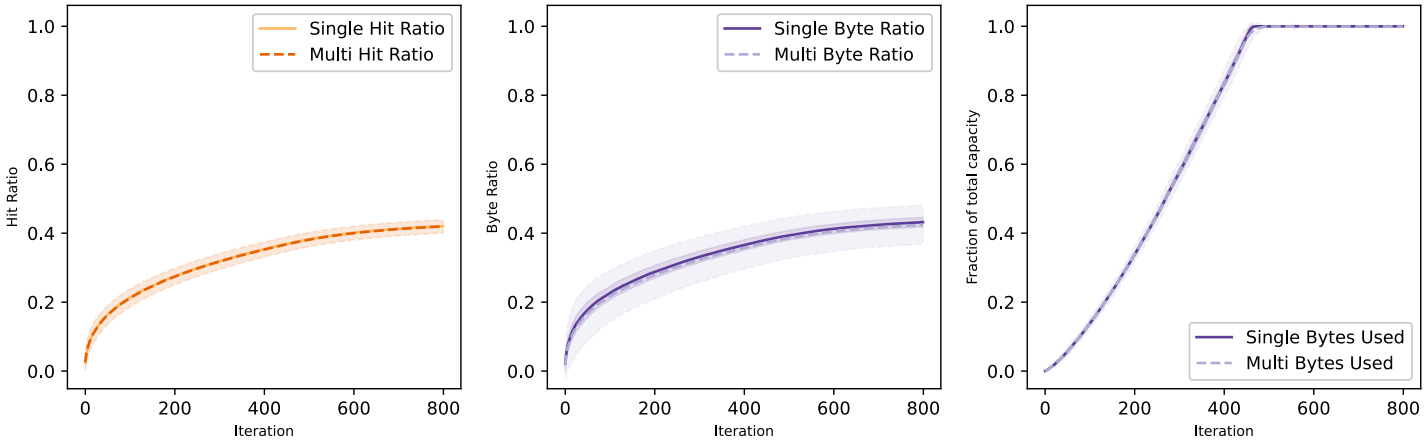
The groups made by the Federated strategy can be seen as a virtual node with the total capacity of all nodes in the group. This sort of abstraction is possible as the Federated strategy shows minimal differences in performance with a larger LRU node that has the same capacity as the network of federated nodes. In this subsection we show, by experiment, that we can indeed reason about the federated approach in this manner.

The main difference between a large LRU node and a network of Federated nodes lies in the global versus local eviction lists. Each node, in a federated group, manages its own eviction list. Because of this, an unequal distribution in resource sizes (the hash only equally distributes the requests) could result in a resource being evicted that would not be evicted on a global eviction list.

In a small experiment we try to measure the significance of this difference. We compare 8 separate nodes with a capacity of 1024MB each to a single node with a capacity of

8192MB. We evaluate this setup with 10 different traces, for 1000 users and 800 iterations, and for each compare the difference between the single- and multi-node setup.

We evaluate the two strategies based on the hit-ratio, bandwidth saved, and storage used over the iterations. We plot these values based on the mean of the different runs and include an error band based on the standard deviation.



(A) *Hit-ratio.* (B) *Fraction of bandwidth saved.* (C) *Fraction of capacity used.*

FIGURE 5.6: *Substitution experiment results*

The experiment results are shown in Figure 5.6. The average hit-ratio (Figure 5.6a), is similar in both setups with a maximum difference of `0.10%`.

The difference in the byte ratio is visible in Figure 5.6b. The error band shows a larger standard deviation for the multi-node setup compared to the single-node setup. We conclude that nodes are indeed affected by the inconsistent distribution of bytes as some perform worse while others perform better. On average both setups perform almost similar with a slightly better performance for the single-node setup, as expected. The maximum difference between the two setups is `0.92%`.

In terms of storage used we find our main difference at the point where both setups reach full capacity. The single-node setup reaches this point some iterations earlier than the multi-node setup, this is visible by the slower growth of the multi-node setup around these iterations in Figure 5.6c. This slowdown is attributed to having reached maximum capacity on some nodes in the multi-setup but not all of them. For additional requests to the nodes at full-capacity there will be no increase in total capacity of the multi-node setup, causing the slowdown visible in the plot.

The differences between the two setups are minimal, we have highlighted where the two differ and by how much. Therefore we argue that we can substitute one for the other with minimal impact on results.

SECTION 6

Evaluation

We perform several evaluations of our proposed strategies to determine the impact of certain changes on the performance. The evaluations allow us to reason about the strong and weak sides of each strategy.

6.1. Cooperative LRU

First we take a look at the Cooperative LRU strategy and the effect of the user trail path `N`. We compare the values of a trail length `1`, `2`, and `3` with our baseline strategy, LRU, and an adaptation of Cooperative LRU strategy that does not make use of mobility data and instead tries to fetch missing content from its direct neighbours.

We run our experiment for the Zipf-0.75, Page-Map, and Zipf-1.30 trace generators outlined in section 3.6.5 for the 14 node setup shown in Figure 4.1. Each cache is provided `1024MiB` of capacity. For each generator we evaluate 10 different traces, each with a different seed. The experiment variables are summarised in Table 6.1.

Table 6.1: *Cooperative LRU Experiment Variables*

Number of users	1000
Number of iterations	5000
Trace Generators	Zipf-0.75, Page-Map, Zipf-1.30
Cache Capacity	1024 MiB
Number of runs	10
Edge Node Setup	14 nodes
Strategies	LRU, Cooperative-LRU(Trail Length=[1, 2, 3]), Neighbouring

6.1.1 Results

The results are shown in Table 6.2. For each experiment we show the *Average Hit Ratio*, *Ratio of Internal Request*, *Average Ratio Bandwidth*, and *Ratio of Internal Bandwidth*. The *Ratio of Internal Requests* highlights the number of internal requests as a fraction of the number of incoming requests, while the *Ratio of Internal Bandwidth* highlights the amount of bytes sent between neighbours compared to the amount of bytes served by the cache.

		Cooperative-LRU			
	LRU	N=1	N=2	N=3	Neighbours
Hit Ratio	0.227±0.011	0.268±0.011	0.279±0.01	0.287±0.011	**0.295±0.019**
Internal Requests	-	0.756±0.006	1.078±0.047	1.374±0.061	1.995±0.528
Bandwidth Saved	0.212±0.016	0.252±0.016	0.263±0.016	0.271±0.016	**0.28±0.022**
Internal Bandwidth	-	0.04±0.006	0.051±0.007	0.059±0.007	0.068±0.011

(A) *ZipF-0.75 Results*

		Cooperative-LRU			
	LRU	N=1	N=2	N=3	Neighbours
Hit Ratio	0.786±0.011	0.864±0.008	0.875±0.007	**0.878±0.007**	0.876±0.01
Internal Requests	-	0.206±0.009	0.264±0.014	0.313±0.015	0.446±0.087
Bandwidth Saved	0.212±0.016	0.252±0.016	0.263±0.016	0.271±0.016	**0.28±0.022**
Internal Bandwidth	-	0.101±0.012	0.114±0.014	0.118±0.014	0.111±0.013

(B) *Page-Map Results*

		Cooperative-LRU			
	LRU	N=1	N=2	N=3	Neighbours
Hit Ratio	0.915±0.006	0.925±0.004	0.927±0.003	0.928±0.003	**0.93±0.004**
Internal Requests	-	0.081±0.005	0.112±0.007	0.14±0.005	0.206±0.043
Bandwidth Saved	0.893±0.028	0.905±0.025	0.908±0.024	0.909±0.024	**0.912±0.023**
Internal Requests	-	0.012±0.005	0.014±0.006	0.016±0.006	0.018±0.006

(C) *ZipF-1.30 Results*

TABLE 6.2: *Cooperative LRU Experiment Results, Tables a-c show the results for the different trace generators. Each table contains, for every setup, the average and standard deviation for the hit ratio, the number of requests made internally (normalised by the number of incoming requests), the fraction of bandwidth saved, and the fraction of the user-facing bandwidth that originates from neighbouring nodes. The best performing setups are highlighted in bold.*

In almost all strategies the neighbouring strategy performs best, at the cost of a higher overhead on the internal network as seen in the *Internal Requests* row. The proposed Cooperative-LRU strategy outperforms the LRU strategy, even when using a trail length of `1`. Longer trail lengths have minimal impact on the performance but bring it close to the performance of the neighbouring strategy while keeping the ratio of *Internal Requests* lower.

This confirms our assumption that using the movement path of the user to direct the node cooperation, results in a more efficient use of network resources at a lower performance impact. It also shows that there is a minimal loss in performance compared to the neighbouring strategy.

For all the following experiments including the Cooperative LRU strategy we will set the *trail length* to `3`.

6.2. Profiles

The size of the profiles play an important role in the profile ranking. The bigger the profile size the more elaborate the ranking will be at the cost of storage space. In this experiment we test 4 orders of magnitude in profile sizes: `10`, `100`, `1000`, and `10000` to determine the effect on the cache performance. In addition larger profile sizes will result in a slower adaptation to new requests made by users as each item in the ranking has the same weight.

The profile size determines how long the request history is that we store for a user. This can be represented as an array of unique identifiers (URIs) stored in a list. We will use the average URI length in our dataset of `74` bytes to calculate the profile sizes representative of the tested values.

Our experiment is evaluated over 10 runs where each run uses a different seed for the traces. The traces are generated by the three trace generators outlined in section 3.6.5. In our experiment we will compare the proposed profile sizes with our baseline LRU strategy using the 14 node setup where each node has a capacity of `1024MiB`. The experiment variables are summarised in Table 6.3.

Table 6.3: *Profiles Experiment Variables*

Number of users	1000
Number of iterations	5000
Trace Generators	Zipf-0.75, Page-Map, Zipf-1.30
Cache Capacity	1024 MiB
Number of runs	10
Edge Node Setup	14 nodes
Strategies	LRU, Profiles(Profile Size=[10, 100, 1000, 10000])

6.2.1 Results

The results are visualised in Table 6.4. In these results we see how the Zipf-0.75 traces, with a more gradual popularity drop-off, require a larger profile size to match the performance of the LRU baseline. The Profiles strategy needs to be able to keep track of more items to achieve similar performance. For the higher profile sizes, `1000` and `10000`, the Profiles strategy is able to outperform the LRU baseline.

The Page-Map and Zipf-1.30 traces have a, comparatively, steeper popularity drop-off. This means there is a smaller subset of files responsible for the majority of incoming requests. We can see the effect of this in our baseline LRU, as it performs significantly better on these two types of traces compared to the Zipf-0.75 traces. It also shows in the high performance at small profile sizes for the Profile Strategy, already outperforming the

		Profiles			
	LRU	N=10	N=100	N=1000	N=10000
Profile Size		740B	7.2KiB	72.3KiB	722.7KiB
Hit Ratio	0.227±0.011	0.167±0.002	0.188±0.005	0.247±0.012	**0.286±0.014**
Internal Requests		0.005±0.001	0.033±0.005	0.13±0.018	0.232±0.044
Bandwidth Saved	0.223±0.032	0.164±0.031	0.185±0.032	0.243±0.033	**0.281±0.031**
Internal Bandwidth		0.002±0.0	0.012±0.001	0.031±0.004	0.04±0.005

(A) *ZipF-0.75 Results*

		Profiles			
	LRU	N=10	N=100	N=1000	N=10000
Profile Size		740B	7.2KiB	72.3KiB	722.7KiB
Hit Ratio	0.788±0.011	0.906±0.004	0.909±0.004	0.915±0.004	**0.927±0.004**
Internal Requests		0.001±0.0	0.004±0.001	0.011±0.002	0.022±0.002
Bandwidth Saved	0.7±0.018	0.874±0.008	0.876±0.008	0.881±0.008	**0.896±0.008**
Internal Bandwidth		0.001±0.0	0.003±0.001	0.007±0.002	0.02±0.004

(B) *Page-Map Results*

		Profiles			
	LRU	N=10	N=100	N=1000	N=10000
Profile Size		740B	7.2KiB	72.3KiB	722.7KiB
Hit Ratio	0.915±0.006	0.931±0.002	0.934±0.002	0.94±0.003	**0.944±0.003**
Internal Requests		0.0±0.0	0.003±0.001	0.009±0.001	0.012±0.002
Bandwidth Saved	0.899±0.037	0.918±0.029	0.921±0.029	0.928±0.026	**0.933±0.024**
Internal Bandwidth		0.0±0.0	0.003±0.001	0.007±0.003	0.008±0.003

(C) *ZipF-1.30 Results*

TABLE 6.4: *Profile Experiment Results, Tables a-c show the results for the different trace generators. Each table contains, for every setup, the average and standard deviation for the hit ratio, the number of requests made internally (normalised by the number of incoming requests), the fraction of bandwidth saved, and the fraction of the user-facing bandwidth that originates from neighbouring nodes. The best performing setups are highlighted in bold.*

baseline LRU from a profile size of `10`, with minor gains in performance for larger sizes.

The chosen profile size will therefore depend on the expected popularity distribution. A more gradual distribution requires larger profile sizes, and thus additional storage to deal with these profiles (depending on the expect amount of users per node).

For all the following experiments including the Profiles strategy we will set the *Profile Size* to `10000`.

6.3. Effects of Storage Capacity

In this experiment we explore the correlation between the storage capacity and cache performance for the proposed strategies. We also include our baseline, LRU, and optimal strategy, Belady's MIN. Both of which do not implement any features for cooperation between the nodes, but are used for comparison.

We experiment with six different storage capacities: `64MiB`, `128MiB`, `256MiB`, `512MiB`, `1024MiB`, and `2048MiB`. Given our resource dataset has a total size of 84Gb, these values represent `0.075%`, `0.15%`, `0.3%`, `0.6%`, `1.2%`, and `2.4%` of the total dataset size.

We evaluate the experiment with traces generated by the three different generators outlined in section 3.6.5. Each trace simulates a 1000 users over a 5000 iterations. The experiment uses the 14 node edge setup outlined in Figure 4.1. We repeat each setup `10` times, each with a trace generated using a different seed, to improve our confidence in the results. The experiment variables are summarised in Table 6.5.

Table 6.5: *Storage Effects Experiment Variables*

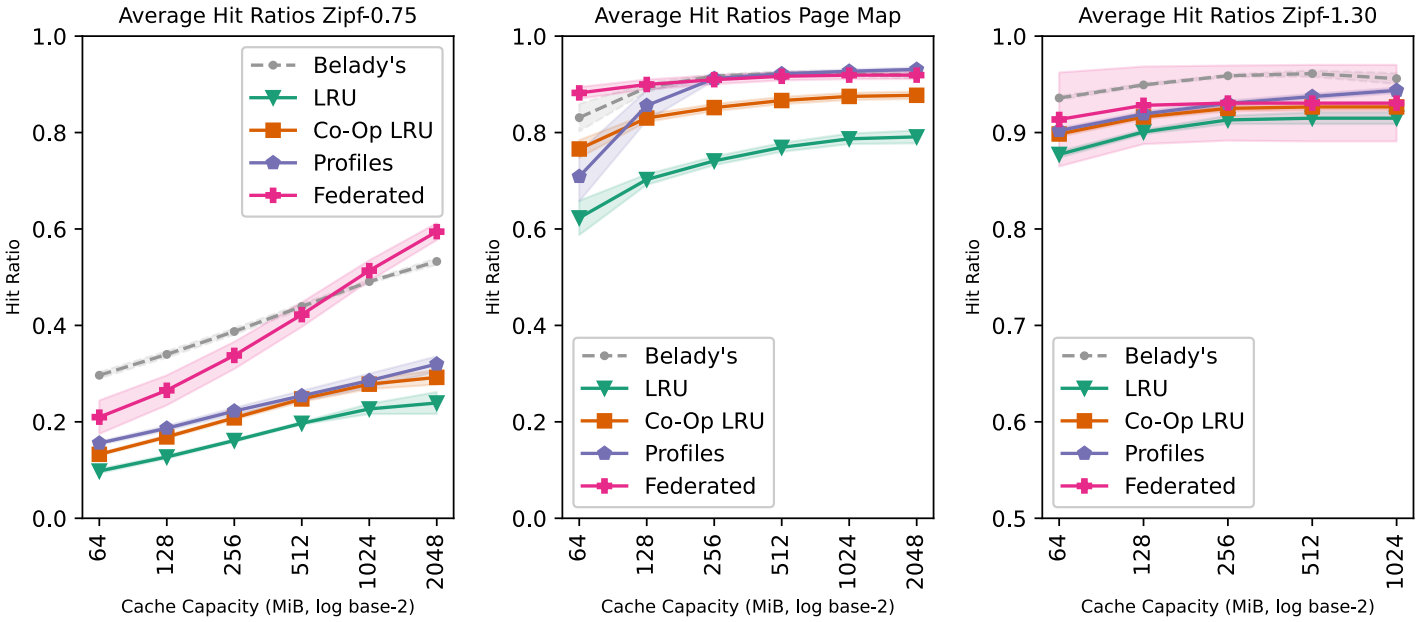
Number of users	1000
Number of iterations	5000
Trace Generators	Zipf-0.75, Page-Map, Zipf-1.30
Cache Capacities	64MiB, 128MiB, 256MiB, 512MiB, 1024MiB, 2048MiB
Number of runs	10
Edge Node Setup	14 nodes
Strategies	Belady's MIN, LRU, Cooperative-LRU(Trail Length=3), Profiles(Profile Size=10000), and Federated

6.3.1 Results

Figure 6.1 shows the average hit ratios achieved by the different proposed strategies, for the different trace generators. Each plot shows the average performance (hit-ratio) on the y-axis while the storage size in bytes is shown on the x-axis. Note that the x-axis uses a log scale with base 2 to match the orders of magnitude in cache sizes. The Zipf-1.30 traces have been given a more restricted y-axis starting at `0.5` instead of `0` to better visualise the differences in strategies.

In addition to this we also leave out the 2048MiB results for the Zipf-1.30 traces as due to the shifted popularity distribution there are not enough different items requested to reliably fill up this cache capacity.

In general the trend visible across the different trace generators and strategies is that with extra storage capacity we are able to achieve better performance. The effect is



(A) *Zipf-0.75 generator* (B) *Page-Map generator* (C) *Zipf-1.30 generator*

FIGURE 6.1: *Effects of Storage results for the different generators. Each plot shows the storage capacity per node versus the average hit ratio achieved.*

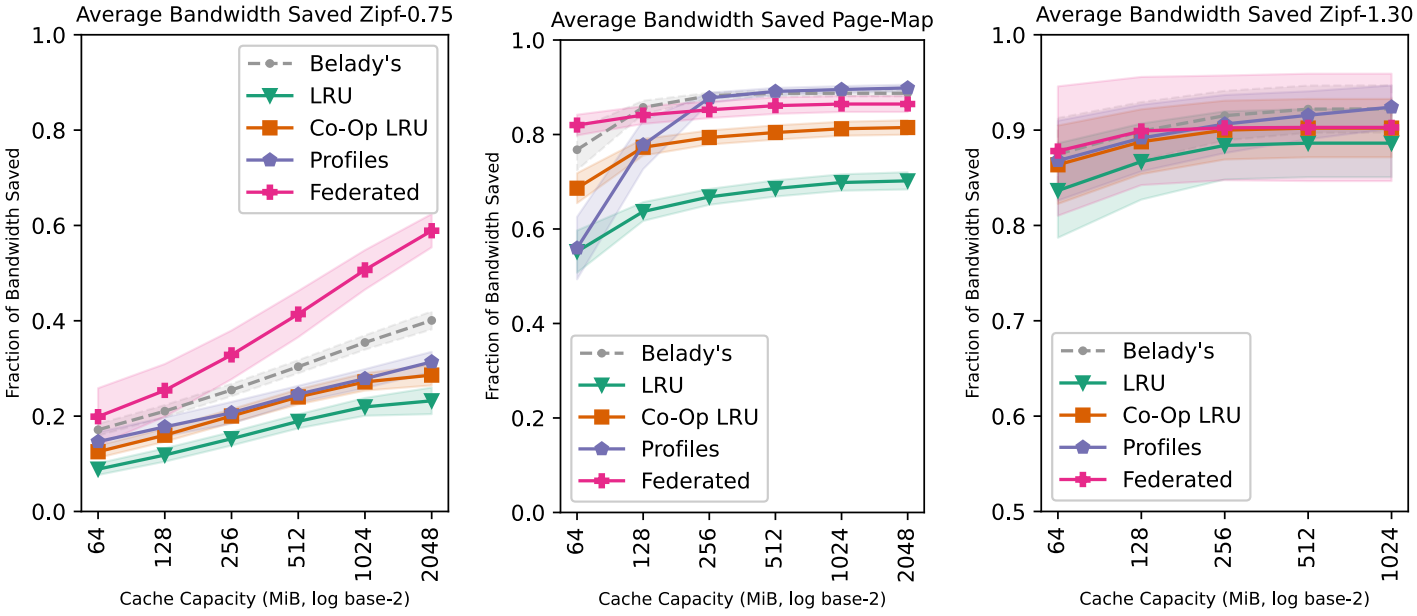
weakest for the Zipf-1.30 traces, as due to the steeper popularity drop-off a small fraction of the resources is responsible for the majority of requests. This is expected as when we look back at the cumulative distribution of requests in section 3.6, we see how with a high exponent for the Zipf-like distribution a large fraction of the requests is generated by a small fraction of very popular content. The same applies to the Page-Map generator where the popularity curve is somewhat less steep. As a result we need very little storage capacity to achieve a reasonable performance, and will see limited impact for additional storage space, for the Page-Map and Zipf-1.30 traces. Still a slight drop-off in performance is visible at the lower end of the cache capacities.

The reverse is true for the Zipf-0.75 traces. Due to the, comparatively, large set of popular resources responsible for the majority of requests, extra storage capacity is utilised to store more of these resources. All strategies therefore perform better with additional storage capacity. Still their performance is on the low end of the scale.

This is the reason the Federated strategy jumps out in terms of performance from all the other strategies in these traces. As the Federated strategy combines the storage capacity, to better utilise the available storage, of all 14 edge nodes it is inline with the performance of a single LRU cache with 14 times the storage size. We even find that due to the better utilisation of the available storage the strategy is able to outperform Belady's MIN (which does not use any cooperation between the nodes).

For the Cooperative LRU strategy we find a mostly consistent improvement over the LRU strategy. This improvement stems from the cooperation between caches to retrieve content from neighbours (which the baseline LRU does not implement) instead of origin.

One of the outliers is the Profiles strategy in the results for the Page-Map generator. Here the strategy sees a significant drop-off in performance with the lowest node size of `64MiB` but performs as one of the best strategies for a node size of `128MiB` and up. The difference here is generated by the inner workings of the profiles method. During profile aggregation the last `N` items are chosen from each profile and aggregated together. When trying to insert a new item, the cache verifies that the item is indeed in the ranking and what items are stored with a lower ranking to evict (in case there is not enough storage available). Thus restricting which resources are allowed in the cache. This restriction, together with the more granular popularity drop-off seen in the Page-Map generator, reduces the performance of the Profiles strategy for lower cache capacities. This effect is not visible in any of the other strategies as they do not maintain a ranking used to determine which items are allowed in the cache. In the results seen for our Page-Map generator this shows that for extremely small capacities the Cooperative LRU strategy is a better choice.



(A) *Zipf-0.75 generator* (B) *Page-Map generator* (C) *Zipf-1.30 generator*

FIGURE 6.2: *Effects of Storage results for the different generators. Each plot shows the storage capacity per node versus the average bandwidth savings achieved.*

In addition to the hit-ratios we also visualised the bandwidth savings achieved by each strategy for the different cache capacities in Figure 6.2. The resulting ratios are lower than the hit-ratios but have the same characteristics. The lower ratios for the bandwidth savings are explained by the trace evaluation performed in section 3.6. Here when we compare the cumulative distribution of requests and the cumulative distribution of bytes, we find that the two do not exactly match and that in our traces the cumulative

distribution of bytes is shifted towards a Zipf-like distribution with a higher exponent than the cumulative distribution of requests. This explains how the more popular items stored in our cache account for more of the total fraction of requests than the total fraction of bytes.

We conclude that the best strategy to apply depends on the cache capacity used and on the characteristics that the incoming requests will have. For requests following a Zipf-like distribution, with a smaller exponent, the federated approach performs best. While for Zipf-1.30 and Page-Map like distributions the Profiles strategy is the best option (for larger cache capacities).

6.4. Effects of Mobility

The Cooperative LRU and Profiles strategies make use of their neighbouring nodes, either for optimizing their storage or to avoid calls to origin. These strategies rely on the assumption that popular content moves with the user. In this experiment we evaluate the effect of the movement speed of users on the performance of these strategies.

We use the mobility speeds: `1-in-2`, `1-in-10`, `1-in-100`, and `1-in-1000` iterations. As the movement is based on a random chance by a uniform random process these speeds represent an average movement speed. These speeds were chosen to reflect an order of magnitude difference, the `1-in-2` is an outlier and chosen over `1-in-1` to give the logic behind the mobility strategies a chance to catch up with the speed of the user.

In this experiment we only look at the traces generated by the *Page-Map* generator as this is the only generator that uses a specialisation on a per-user basis. Without a specialisation, such as in the *Zipf* Generator the popularity of different resources will be too similar across users. This means there will be no difference in resource popularity per user.

As the movement speed is part of the generated trace we generate a new trace for each different mobility speed. To allow for maximum movement we use the 14 node setup outlined in Figure 4.1. We repeat each setup in the experiment `10` times, with a trace generated with a different seed, to improve our confidence in the results. The experiment variables are summarised in Table 6.6.

6.4.1 Results

The results of our experiment are shown in Figure 6.3 and Figure 6.4a. In each plot we visualised the user speed by the average number of iterations between user movement on the x-axis (in log-scale to match the orders-of-magnitude differences) and one of the performance metrics on the y-axis.

As the Federated strategy does not consider mobility it shows a constant performance for all movement speeds in both hit ratio and bandwidth savings. This also means the performance achieved by the Federated strategy is among the highest. The drawback, however, is visible in Figure 6.3c and Figure 6.4a where we plot the number of requests

Table 6.6: *Mobility Effects Experiment Variables*

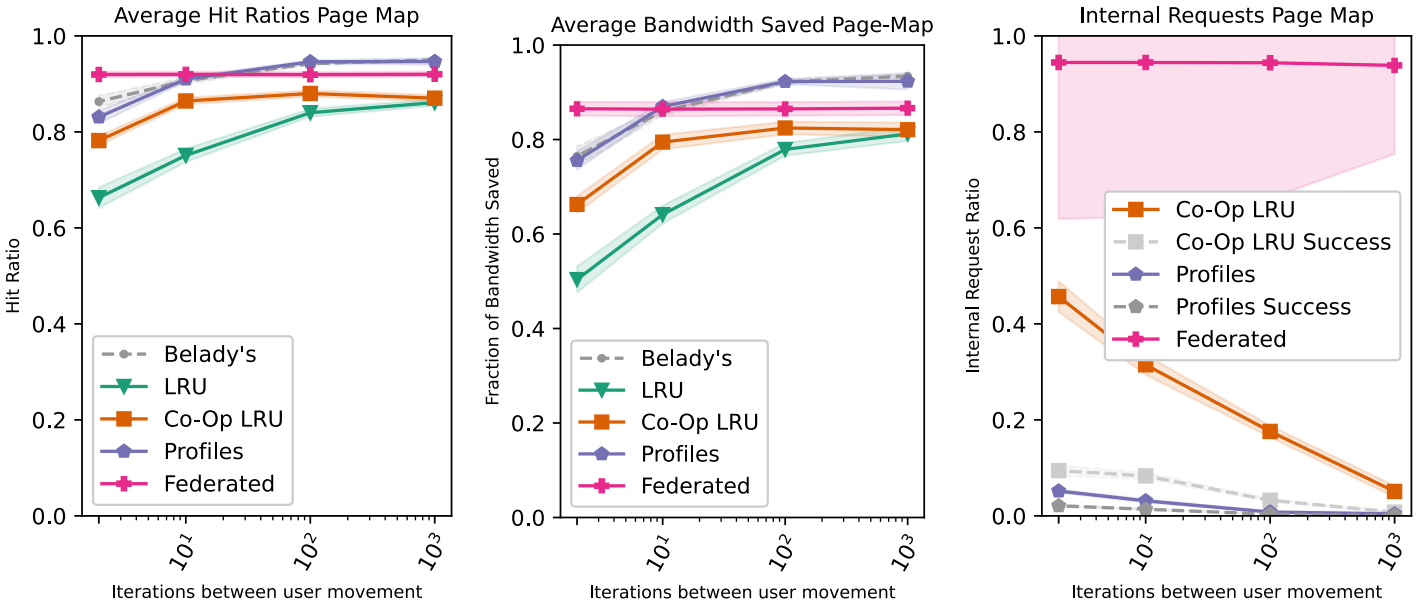
Number of users	1000
Number of iterations	5000
Trace Generators	Page-Map
Mobility Speeds	1-in-2, 1-in-10, 1-in-100, 1-in-1000 iterations
Cache Capacity	1024MiB
Number of runs	10
Edge Node Setup	14 nodes
Strategies	Belady's MIN, LRU, Cooperative-LRU(Trail Length=3), Profiles(Profile Size=10000), and Federated

made internally as a ratio to the amount of incoming requests and show the internal bandwidth usage normalised by the user-facing bandwidth. For a 14 node setup the federated strategy on average forwards `94.3%` of the requests and `94.5%` of the bandwidth. This is in line with the expected `92.9%` of requests being forwarded. The more edge nodes are grouped together in a Federated group the more we rely on the internal links between the nodes. Compared to the other strategies the Federated approach uses a significant amount more of the internal bandwidth in our 14 node setup.

All other strategies see a degradation in performance as the user speed increases. This effect is explained by the specialisations and different popularity rankings for each user. The faster a user moves the less time each cache node has to adapt to the popularity of its connected users. Together with the fact that each user has their own specialisation we create a scenario where it is more likely that a specific item has not yet been cached on the newly connected node. Therefore reducing the hit-rate for Belady's MIN, LRU, Cooperative-LRU and Profiles strategies as seen in Figure 6.3a.

The baseline LRU strategy is the most affected while the Cooperative LRU strategy is able to compensate for the loss in performance using internal requests. However as seen in Figure 6.3c the increase in internal requests for the Cooperative LRU strategy is not met with a higher success ratio. This is one of the effects inherent to the Cooperative LRU strategy, as the difficulty to find the correct resource popularity increases, for example by fast moving users, it uses more internal requests to try and find the requested resources from neighbours. But due to the fast movement speed it is unlikely that the neighbouring nodes contain the requested resource, hence there is no increase in the number of successful internal requests.

The Profiles strategy performs the best in this scenario, except for the fasted moving speed, where the federated approach performs better. This shows that for very dense edge networks, where users are likely to switch between nodes often, the federated approach is the better option. This matches with the target use case for the Federated strategy, as it should be applied to edge nodes that are close together and have a reliable connection between them. For example an edge network within a city. However for movement over larger distances the profiles strategy performs better without the high overhead in internal traffic. This matches with a more sparse edge network, for example where each city in a



(A) *Average Hit Ratios* (B) *Average Bandwidth saved* (C) *Ratio of internal requests*

FIGURE 6.3: *Effects of Mobility experiment results. The plots show, from left to right, the user movement speed versus the hit-ratio, fraction of bandwidth saved, and requests forwarded internally.*

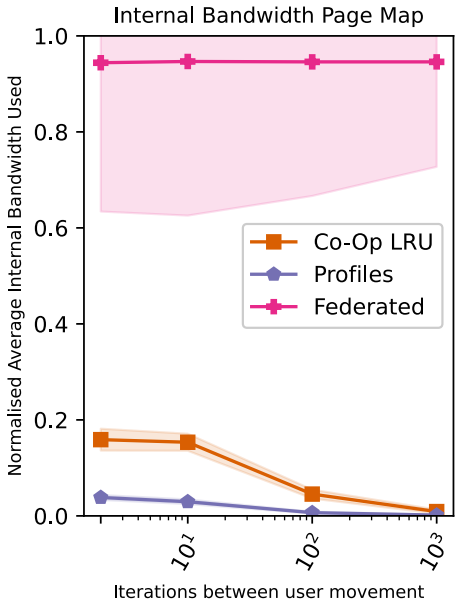
specific area has one edge node.

6.5. Effects of Network Density

Similar to our experiment on the limited population (see section 4.3) we evaluate the performance of the proposed strategies on different edge network setups. In this experiment we try to evaluate whether these strategies reduce the decreased performance seen by edge caches with limited popularity.

In our experiment we evaluate the performance of the Cooperative-LRU and Profiles strategies over the four edge node setups outlined in Figure 4.1. Based on our results in the section 4.3 we will decrease the node capacity on larger edge setups to keep a total capacity the same, as in this scenario the effect was strongest. We have chosen the total capacity of `3072MiB` to provide an equal playing field to the Profiles strategy. In the Experiment on Storage Effects (section 6.3) we have seen how the Profiles strategy is sensitive to storage capacities lower than `128MiB` in the 14 edge node setup. With a capacity of `3072MiB`, each node in the 14 node setup will have access to `219MiB`. We focus on the traces generated by the Page-Map generator as the effect of a limited population was the strongest in this scenario.

In addition we show the results of the Profile strategy for a profile size of `100` and `10000`. As due to the uniform distribution of specialisations, the large amount of users, and slow adaptation of the profile history on the single node setup, the ranking will move



(A) *Average bandwidth used internally, normalised by total outgoing bandwidth*

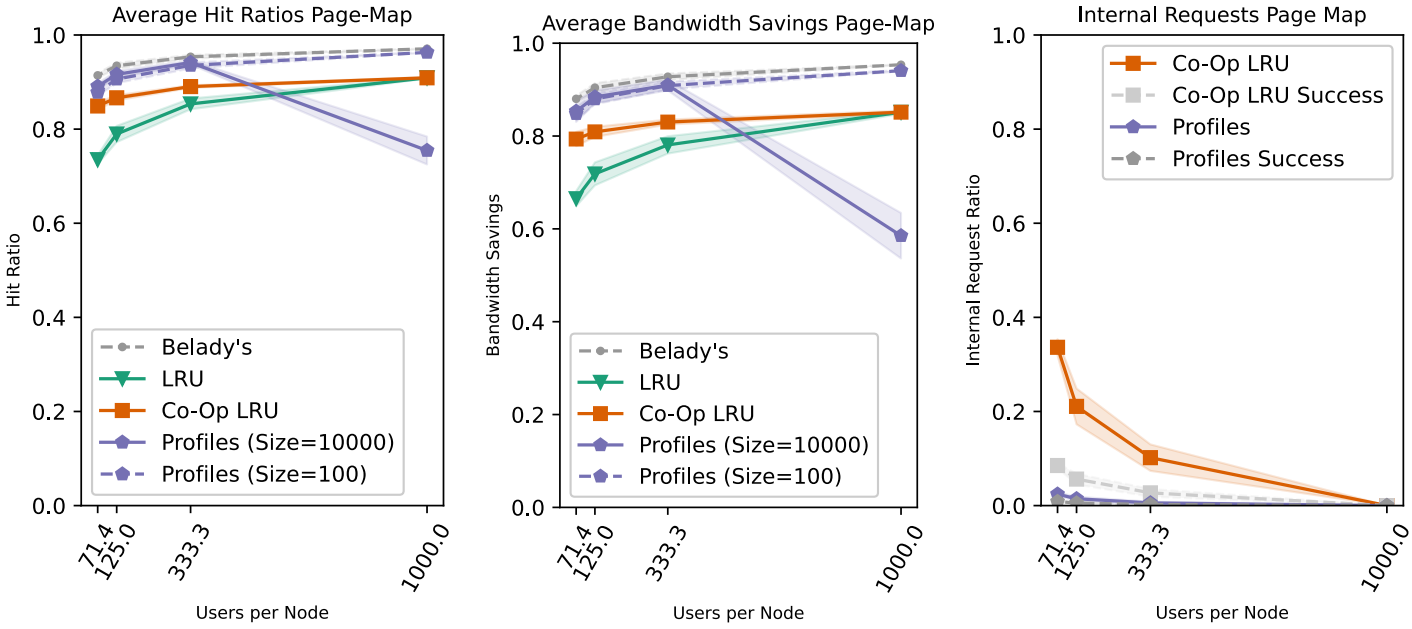
FIGURE 6.4: *Effects of Mobility experiment showing the internal bandwidth usage normalised by the total outgoing (user-facing) bandwidth.*

towards a uniform distribution, decreasing performance. A smaller profile size, in relation to the number of iterations, is better able to adapt to the changes in popularity for each user and will therefore show better performance when users all users are grouped together.

The experiment variables are summarised in Table 6.7.

TABLE 6.7: *Density Effects Experiment Variables*

Number of users	1000
Number of iterations	5000
Trace Generator	Page-Map
Cache Capacities	219-3072MiB (3072MiB total)
Number of runs	8
Edge Node Setups	1, 3, 8, 14 nodes
Strategies	Belady's MIN, LRU, Cooperative-LRU(Trail Length=3), Profiles(Profile Size=[10000, 100])



(A) *Average hit ratio.* (B) *Average bandwidth savings.* (C) *Ratio of internal requests.*

FIGURE 6.5: *Density Experiment results for the Page-Map generator. Each plot shows the average number of users per node versus, from left-to-right, the hit ratio, fraction of bandwidth saved, and fraction of requests forwarded between neighbours (including the fraction of successful responses).*

6.5.1 Results

We plot our results in Figure 6.5. Here the number of users per node are shown on the x-axis while different performance metrics are shown on the y-axis for each plot. For comparison we included the results for LRU and Belady's MIN from the limited population experiment in section 4.3.

We find that for the Cooperative LRU strategy, while the impact is weaker than for the baseline LRU strategy, there is still an impact on performance from the reduced population. This shows that by cooperation with neighbouring nodes we are somewhat able to reduce the negative performance effect of the reduced population.

Our Profiles strategy similarly shows a reduced reduction in performance for the different population sizes (if we consider a lower profile size in the single node setup). Overall the strategy performs close to the optimal strategy without the internal overhead seen in the Cooperative LRU strategy.

In Figure 6.5c we compare the different population densities per node to the ratio of incoming requests forwarded between neighbours. Here the Cooperative LRU shows an increase in the number of requests forwarded as the network increases in node count. The Profiles strategy only checks the last known nodes for misses on resources that are included in the ranking. This results in less requests forwarded internally, yet at the same time the Profiles method is able to achieve a higher hit-ratio and backhaul-savings than

the Cooperative LRU strategy.

Based on these two findings we conclude that, by moving profiles with the users the Profiles strategy is better able to estimate what content is popular for its users compared to the Cooperative LRU strategy. Because of this it is able to make a better estimation on what content to keep in the cache or to evict. However the Cooperative LRU strategy shows a more gradual drop-off in performance when the user population per node decreases.

6.6. Hybrid Strategies

Now that we have explored some of the characteristics of our proposed strategies, we perform a final experiment where we combine the mobility strategies with the federated approach to create a hybrid strategy. In this strategy we apply the logic of one of the mobility strategies on a virtual node that is created by combining multiple smaller nodes using the Federated strategy. In addition to the mobility strategies, Cooperative LRU and Profiles, we also apply our baseline LRU to these virtual nodes for comparison.

For this experiment we make use of the 14 node setup outlined in Figure 4.1. These 14 nodes will be grouped together into three virtual nodes using the Federated strategy as outlined in Figure 6.6. We showed in section 5.5.1 that this simplification into virtual nodes has minimal impact on the actual performance. Virtual nodes 1 and 3 have 5 physical nodes while virtual node 2 has only 4. We set the total storage capacity to `2048MiB` which gets divided uniformly over all edge nodes. This means that the virtual nodes 1, 2, and 3 have a cache capacity of `731MiB`, `585MiB`, and `731MiB` respectively.

FIGURE 6.6: *The 14 node setup divided into 3 virtual nodes.*

In our evaluations we use the traces generated by the three different trace generators as outlined in section 3.6.5 for a 14 node setup. During the evaluation we map the requests for these 14 nodes to the virtual nodes as outlined in Figure 6.6. We repeat

this experiment 8 times using traces with different seeds. The experiment variables are summarised in Table 6.8.

TABLE 6.8: *Hybrid Experiment Variables*

Number of users	1000
Number of iterations	5000
Trace Generator	Zipf-0.75, Page-Map, Zipf-1.30
Cache Capacities	585-731MiB (2048MiB total)
Number of runs	8
Edge Node Setup	14 nodes (3 virtual nodes)
Strategies	Hybrid LRU, Hybrid Cooperative-LRU(Trail Length=3), Hybrid Profiles(Profile Size=10000)

6.6.1 Results

	14 Node Co-Op LRU	Profiles	Hybrid LRU	Co-Op LRU	Profiles
Hit Ratio	0.169±0.003	0.188±0.005	0.205±0.005	0.233±0.003	**0.286±0.007**
Bandwidth Saved	0.16±0.014	0.196±0.004	0.198±0.018	0.226±0.017	**0.275±0.018**

(A) *ZipF-0.75 Results*

	14 Node Co-Op LRU	Profiles	Hybrid LRU	Co-Op LRU	Profiles
Hit Ratio	0.83±0.008	0.857±0.032	0.849±0.007	0.883±0.004	**0.925±0.008**
Bandwidth Saved	0.773±0.018	0.78±0.054	0.785±0.008	0.825±0.006	**0.882±0.016**

(B) *Page-Map Results*

	14 Node Co-Op LRU	Profiles	Hybrid LRU	Co-Op LRU	Profiles
Hit Ratio	0.916±0.002	0.919±0.002	0.942±0.003	0.946±0.002	**0.948±0.002**
Bandwidth Saved	0.888±0.034	0.894±0.044	0.919±0.029	0.924±0.027	**0.927±0.026**

(C) *ZipF-1.30 Results*

TABLE 6.9: *Hybrid Experiment Results, Tables a-c show the results for the different trace generators. Each table contains, for every setup, the average and standard deviation for the hit ratio and the fraction of bandwidth saved. The best performing setups are highlighted in bold.*

The results of our experiment for the different trace generators are shown in Table 6.9. We have included the results for the non-hybrid, 14 node setup with a capacity of `128MiB` per node (compared to `146MiB` in the hybrid setup), for both the Cooperative LRU and Profiles strategies from the Experiment on Storage Capacity. These results allow us to compare the same setup with and without hybrid strategy applied.

For all traces we find that the Cooperative LRU and Profiles strategy perform better in the Hybrid approach (using 3 virtual nodes) than in their non-hybrid counterpart (using 14 nodes). We compare this result with the findings of the Experiment on Mobility (see section 6.4). As we group nodes together using the Federated strategy we create a set of virtual nodes that see less movement between them, as they are made up of physically larger areas. In our Experiment on Mobility we showed that less movement has a positive effect on cache performance, just as seen in the results for the Hybrid approaches.

In general the hybrid strategy using the Profiles strategy performs the best, for all trace generators. This is inline with the results for the non-hybrid variations.

6.7. Mitigating the effects of a limited population

With these results we can answer **RQ3**: "*How can cooperation with neighbouring nodes decrease the effects of a limited population?*" and **RQ4**: "*What are the trade-offs and limitations of the proposed strategies?*". We proposed three strategies to reduce the impact of a limited population on the cache performance in section 5. In this chapter we evaluated the different strategies and found how each of them was able to reduce the effects of a limited population.

The Cooperative LRU strategy reduces the performance impact caused by the limited population through cooperation upon request misses. It does this at the cost of of internal requests that increase with a longer *node-trail* or for more gradual popularity distributions.

The Federated strategy combines nodes together into a virtual larger node and therefore eliminates the issue altogether as the virtual nodes serve the entire population of the edge network. The main limitation is that the strategy relies heavily on internal communication between these nodes, and therefore does not scale easily to larger or more distributed edge networks.

The Profiles strategy, by moving popularity data with the users, is also able to reduce the effects of a limited population, by moving popularity data with the user. In addition it does not rely on heavy internal communication like the Federated strategy. Out of the three it is the most complex strategy and requires the use of state in the form of a resource ranking and set of connected profiles per node. Additionally larger profile sizes come with better performance at the cost of additional storage space for each profile.

Finally we combined the Federated strategy with the two mobility strategies to achieve better performance in the Hybrid approach as seen in section 6.6. This performance boost is related to the artificial slowdown in user movement created by the grouping of physical nodes into a virtual node with a larger physical area. This plays into the results seen in the Experiment on Mobility Effects where the Federated strategy, by design, sees no

impact while the mobility strategies are able to achieve a better performance for slower user movement.

While we cannot be sure on what scale we will see edge nodes in the near future, we can speculate what our results would mean for a situation where at least one edge node per municipality exists. The Netherlands has a total of 355 municipalities[11] and counts over 3000 5G base stations (per mobile provider)[8] for an average of 1 edge node per 8 base stations (or 1 per 24 if the mobile operators would work together).

For a mobile network that operates on a national scale this means that a potential setup using the Hybrid approach could be to combine edge nodes, in smaller areas such as within cities or between smaller towns, together into virtual nodes using the Federated Strategy. This would increase the population per virtual node as well as the total available capacity. This will already reduce the effect of the limited population as seen in section 6.3 and section 6.5. In addition by applying one of the mobility strategies over these virtual nodes we allow content and popularity data to move with the users. The Cooperative-LRU strategy allows content to move with the users but does rely on the internal bandwidth between edge nodes. By design the Profiles strategy will rank the content that is popular for each virtual node, like city or region, based on the connected Profiles. Thus adapting to the popularity for the local population it changes, for example because people move between their homes and work, or because a large event attracts more and different people to a specific area.

[11] opendata.cbs.nl/statline/#/CBS/nl/dataset/70072ned/table (September 24, 2021)

SECTION 7

Conclusion

We started our work by introducing one of the problems faced by mobile networks at this time. The increase in mobile IP traffic poses serious challenges for the mobile network [34, 15]. Edge caching could help reduce the strain on mobile networks by storing popular content closer to the user [18]. While previous works explore the idea of a mobile- or edge-CDN for this purpose they find that small user populations, for reactive cache nodes, significantly reduce the achieved performance [11, 17, 21].

We confirmed that this effect is visible in our simulation and evaluated three different strategies, Cooperative LRU, Profiles, and Federated, to help reduce this negative performance impact at edge-nodes. In our evaluation, we show that both the Cooperative LRU and Profiles strategies are able to reduce the negative effects of a limited population through cooperation and by moving popularity data with the users.

We show that the Federated approach is costly in terms of internal overhead and bandwidth and therefore limited to smaller scale edge nodes. However, in our Hybrid approach we pair groups of edge nodes together into virtual nodes using the Federated strategy and apply one of the two mobility strategies on these virtual nodes. This hybrid approach combines the strengths of the different strategies, resulting in an additional performance improvement.

For mobile networks our work shows that, even for different types of content popularity distributions, a reduction in backhaul traffic can be achieved with the use of a local CDN. By incorporating mobility in their caching solutions we can achieve better performance and reduce the negative impact of the reduced population per node seen in edge caches. This would reduce the need for expensive network upgrades and improve the overall efficiency of the backhaul use. This makes edge caching an additional incentive for mobile operators to invest in edge computing.

In addition we have shown several characteristics of our proposed strategies such as how they react to different configuration, storage capacities, user speeds, and what the associated costs are in internal traffic. We hope that these characteristics together with the open source simulation and experiments provide a foundation for future work in this direction.

References

[1] Nasreen Anjum et al. "Device-To-Device (D2D) Communication As a Bootstrapping System in a Wireless Cellular Network". In: *IEEE Access* 7 (2019), pp. 6661–6678. DOI: 10.1109/access.2019.2890987. URL: https://doi.org/10.1109/access.2019.2890987.

[2] Jean-Philippe Aumasson and Daniel J Bernstein. "SipHash: a fast short-input PRF". In: *International Conference on Cryptology in India*. Springer. 2012, pp. 489–508.

[3] Henri Bal et al. "A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term". In: *Computer* 49.5 (2016), pp. 54–63. DOI: 10.1109/mc.2016.127. URL: https://doi.org/10.1109/mc.2016.127.

[4] Ejder Bastug, Mehdi Bennis, and Mérouane Debbah. "Living on the Edge: the Role of Proactive Caching in 5g Wireless Networks". In: *IEEE Communications Magazine* 52.8 (2014), pp. 82–89. DOI: 10.1109/mcom.2014.6871674. URL: https://doi.org/10.1109/mcom.2014.6871674.

[5] Ejder Bastug et al. "Big Data Meets Telcos: a Proactive Caching Perspective". In: *Journal of Communications and Networks* 17.6 (2015), pp. 549–557. DOI: 10.1109/jcn.2015.000102. URL: https://doi.org/10.1109/jcn.2015.000102.

[6] L. A. Belady. "A Study of Replacement Algorithms for a Virtual-Storage Computer". In: *IBM Systems Journal* 5.2 (1966), pp. 78–101. DOI: 10.1147/sj.52.0078. URL: https://doi.org/10.1147/sj.52.0078.

[7] L. Breslau et al. "Web caching and Zipf-like distributions: evidence and implications". In: *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*. - 1999. DOI: 10.1109/infcom.1999.749260. URL: https://doi.org/10.1109/infcom.1999.749260.

[8] Meeyoung Cha et al. "I tube, you tube, everybody tubes". In: *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement - IMC '07*. - 2007. DOI: 10.1145/1298306.1298309. URL: https://doi.org/10.1145/1298306.1298309.

[9] Min Chen et al. "Mobility-Aware Caching and Computation Offloading in 5g Ultra-Dense Cellular Networks". In: *Sensors* 16.7 (2016), p. 974. DOI: 10.3390/s16070974. URL: https://doi.org/10.3390/s16070974.

[10] Cisco. *Cisco annual internet report (2018–2023) white paper*. 2020. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (visited on 05/03/2021).

[11] Salah-Eddine Elayoubi and James Roberts. "Performance and Cost Effectiveness of Caching in Mobile Access Networks". In: *Proceedings of the 2nd ACM Conference on Information-Centric Networking*. Sept. 2015. DOI: 10.1145/2810156.2810168. URL: https://doi.org/10.1145/2810156.2810168.

[12] J Erman et al. "To Cache Or Not To Cache: the 3g Case". In: *IEEE Internet Computing* 15.2 (2011), pp. 27–34. DOI: 10.1109/mic.2010.154. URL: https://doi.org/10.1109/mic.2010.154.

[13] Abderrahime Filali et al. "Multi-Access Edge Computing: a Survey". In: *IEEE Access* 8 (2020), pp. 197017–197046. DOI: 10.1109/access.2020.3034136. URL: https://doi.org/10.1109/access.2020.3034136.

[14] GMDT Forecast. "Cisco visual networking index: global mobile data traffic forecast update, 2017–2022". In: *Update* 2017 (2019), p. 2022.

[15] Fabio Giust et al. "Caching in Flat Mobile Networks: Design and Experimental Analysis". In: *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*. May 2015. DOI: 10.1109/vtcspring.2015.7145722. URL: https://doi.org/10.1109/vtcspring.2015.7145722.

[16] Yun Chao Hu et al. "Mobile edge computing—A key technology towards 5G". In: *ETSI white paper* 11.11 (2015), pp. 1–16.

[17] Mathieu Leconte et al. "Placing dynamic content in caches with small population". In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. Apr. 2016. DOI: 10.1109/infocom.2016.7524380. URL: https://doi.org/10.1109/infocom.2016.7524380.

[18] Chao Li et al. "Edge-Oriented Computing Paradigms". In: *ACM Computing Surveys* 51.2 (2018), pp. 1–34. DOI: 10.1145/3154815. URL: https://doi.org/10.1145/3154815.

[19] Liying Li, Guodong Zhao, and Rick S. Blum. "A Survey of Caching Techniques in Cellular Networks: Research Issues and Challenges in Content Placement and Delivery Strategies". In: *IEEE Communications Surveys & Tutorials* 20.3 (2018), pp. 1710–1732. DOI: 10.1109/comst.2018.2820021. URL: https://doi.org/10.1109/comst.2018.2820021.

[20] Jinjin Liang et al. "When HTTPS Meets CDN: A Case of Authentication in Delegated Service". In: *2014 IEEE Symposium on Security and Privacy*. May 2014. DOI: 10.1109/sp.2014.12. URL: https://doi.org/10.1109/sp.2014.12.

[21] Dong Liu et al. "Caching At the Wireless Edge: Design Aspects, Challenges, and Future Directions". In: *IEEE Communications Magazine* 54.9 (2016), pp. 22–28. DOI: 10.1109/mcom.2016.7565183. URL: https://doi.org/10.1109/mcom.2016.7565183.

[22] Francesco Malandrino, Carla Chiasserini, and Scott Kirkpatrick. "The price of fog". In: *Proceedings of the First International Workshop on Internet of Vehicles and Vehicles of Internet - IoV-VoI '16*. - 2016. DOI: 10.1145/2938681.2938682. URL: https://doi.org/10.1145/2938681.2938682.

[23] Seok-Hwan Park, Osvaldo Simeone, and Shlomo Shamai Shitz. "Joint Optimization of Cloud and Edge Processing for Fog Radio Access Networks". In: *IEEE Transactions on Wireless Communications* 15.11 (2016), pp. 7621–7632. DOI: 10.1109/twc.2016.2605104. URL: https://doi.org/10.1109/twc.2016.2605104.

[24] Georgios Paschos et al. “Wireless Caching: Technical Misconceptions and Business Barriers”. In: *IEEE Communications Magazine* 54.8 (2016), pp. 16–22. DOI: 10.1109/mcom.2016.7537172. URL: https://doi.org/10.1109/mcom.2016.7537172.

[25] Andrea Passarella. “A Survey on Content-Centric Technologies for the Current Internet: Cdn and P2p Solutions”. In: *Computer Communications* 35.1 (2012), pp. 1–32. DOI: 10.1016/j.comcom.2011.10.005. URL: https://doi.org/10.1016/j.comcom.2011.10.005.

[26] Lakshmish Ramaswamy, Ling Liu, and Arun Iyengar. “Scalable Delivery of Dynamic Content Using a Cooperative Edge Cache Grid”. In: *IEEE Transactions on Knowledge and Data Engineering* 19.5 (2007), pp. 614–630. DOI: 10.1109/tkde.2007.1031. URL: https://doi.org/10.1109/tkde.2007.1031.

[27] James Roberts and Nada Sbihi. “Exploring the memory-bandwidth tradeoff in an information-centric network”. In: *Proceedings of the 2013 25th International Teletraffic Congress (ITC)*. Sept. 2013. DOI: 10.1109/itc.2013.6662936. URL: https://doi.org/10.1109/itc.2013.6662936.

[28] Ion Stoica et al. “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications”. In: *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM ’01. San Diego, California, USA: Association for Computing Machinery, 2001, pp. 149–160. ISBN: 1581134118. DOI: 10.1145/383059.383071. URL: https://doi.org/10.1145/383059.383071.

[29] Tarik Taleb and Adlen Ksentini. “Follow Me Cloud: Interworking Federated Clouds and Distributed Mobile Networks”. In: *IEEE Network* 27.5 (2013), pp. 12–19. DOI: 10.1109/mnet.2013.6616110. URL: https://doi.org/10.1109/mnet.2013.6616110.

[30] Shangguang Wang et al. “A Survey on Service Migration in Mobile Edge Computing”. In: *IEEE Access* 6 (2018), pp. 23511–23528. DOI: 10.1109/access.2018.2828102. URL: https://doi.org/10.1109/access.2018.2828102.

[31] Xiaofei Wang et al. “A Survey of Green Mobile Networks: Opportunities and Challenges”. In: *Mobile Networks and Applications* 17.1 (2011), pp. 4–20. DOI: 10.1007/s11036-011-0316-4. URL: https://doi.org/10.1007/s11036-011-0316-4.

[32] Xiaofei Wang et al. “Cache in the Air: Exploiting Content Caching and Delivery Techniques for 5g Systems”. In: *IEEE Communications Magazine* 52.2 (2014), pp. 131–139. DOI: 10.1109/mcom.2014.6736753. URL: https://doi.org/10.1109/mcom.2014.6736753.

[33] Shinae Woo et al. “Comparison of caching strategies in modern cellular backhaul networks”. In: *Proceeding of the 11th annual international conference on Mobile systems, applications, and services - MobiSys ’13*. - 2013. DOI: 10.1145/2462456.2464442. URL: https://doi.org/10.1145/2462456.2464442.

[34] Junfeng Xie et al. “Icicd: an Efficient Content Distribution Architecture in Mobile Cellular Network”. In: *IEEE Access* 5 (2017), pp. 3205–3215. DOI: 10.1109/access.2017.2671745. URL: https://doi.org/10.1109/access.2017.2671745.

[35] Dongzhu Xu et al. "Understanding Operational 5G: A First Measurement Study on Its Coverage, Performance and Energy Consumption". In: *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '20: Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication. Virtual Event USA: ACM, July 30, 2020, pp. 479–494. ISBN: 978-1-4503-7955-7. DOI: 10.1145/3387514.3405882. URL: https://doi.org/10.1145/3387514.3405882 (visited on 11/11/2020).

[36] Chenchen Yang et al. "Analysis on Cache-Enabled Wireless Heterogeneous Networks". In: *IEEE Transactions on Wireless Communications* 15.1 (2016), pp. 131–145. DOI: 10.1109/twc.2015.2468220. URL: https://doi.org/10.1109/twc.2015.2468220.

[37] F. Z. Yousaf et al. "Mobile Cdn Enhancements for Qoe-Improved Content Delivery in Mobile Operator Networks". In: *IEEE Network* 27.2 (2013), pp. 14–21. DOI: 10.1109/mnet.2013.6485091. URL: https://doi.org/10.1109/mnet.2013.6485091.

[38] Engin Zeydan et al. "Big Data Caching for Networking: Moving From Cloud To Edge". In: *IEEE Communications Magazine* 54.9 (2016), pp. 36–42. DOI: 10.1109/mcom.2016.7565185. URL: https://doi.org/10.1109/mcom.2016.7565185.

[39] Meng Zhang, Hongbin Luo, and Hongke Zhang. "A Survey of Caching Mechanisms in Information-Centric Networking". In: *IEEE Communications Surveys & Tutorials* 17.3 (2015), pp. 1473–1499. DOI: 10.1109/comst.2015.2420097. URL: https://doi.org/10.1109/comst.2015.2420097.