EdgeCut: Fast and Low-overhead Access of User-associated Contents from Edge Servers

Yi Liu
University of California Santa Cruz
yliu634@ucsc.edu

Minmei Wang
University of Connecticut

Shouqian Shi
University of California Santa Cruz

Yang Wang
The Ohio State University

Chen Qian
University of California Santa Cruz
cqian12@ucsc.edu

ABSTRACT

User-associated contents play an increasingly important role in modern network applications. With growing deployments of edge servers, the capacity of content storage in edge clusters significantly increases, which provides great potential to satisfy content requests with much shorter latency. However, the large number of contents also causes the difficulty of searching contents on edge servers in different locations because indexing contents costs huge DRAM on each edge server. In this work, we explore the opportunity of efficiently indexing user-associated contents and propose a scalable content-sharing mechanism for edge servers, called EdgeCut, that significantly reduces content access latency by allowing many edge servers to share their cached contents. We design a compact and dynamic data structure called Ludo Locator that returns the IP address of the edge server that stores the requested user-associated content. We have implemented a prototype of EdgeCut in a real network environment running in a public geo-distributed cloud. The experiment results show that EdgeCut reduces content access latency by up to 50% and reduces cloud traffic by up to 50% compared to existing solutions. The memory cost is less than 50MB for 10 million mobile users. The simulations using real network latency data show EdgeCut's advantages over existing solutions on a large scale.

CCS CONCEPTS

• **Networks** → *Location based services.*

KEYWORDS

Edge computing; Edge location service; User-associated data

ACM Reference Format:
Yi Liu, Minmei Wang, Shouqian Shi, Yang Wang, and Chen Qian. 2023. EdgeCut: Fast and Low-overhead Access of User-associated Contents from Edge Servers. In *The Eighth ACM/IEEE Symposium on Edge Computing (SEC '23), December 6–9, 2023, Wilmington, DE, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3583740.3628439

SEC '23, December 6–9, 2023, Wilmington, DE, USA

ACM ISBN 979-8-4007-0123-8/23/12…$15.00
https://doi.org/10.1145/3583740.3628439

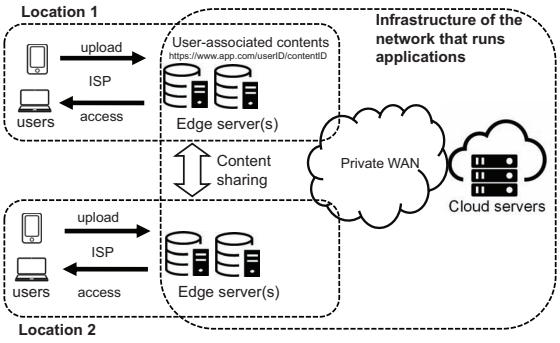


Figure 1: Infrastructure of edge content caching.

1 INTRODUCTION

User-associated contents have been playing an increasingly important role in modern network applications. Each user-associated content is defined as *a file directly related to an application user.* The user profiles, posts, and uploaded photos of online social networks such as Facebook [9] and Twitter [39], documents of file synchronization services such as Google Drive and Dropbox, and the videos of video-sharing applications such as TikTok are all associated with specific users and shared with other accounts interacting with these users.

Content caching methods, such as web caches, content delivery networks (CDNs) [21, 31], and edge service [27, 40], are widely recognized as win-win solutions that reduce both service latency to users and the traffic overhead of network servers and data centers. For a local community, such as all users in a city, a content provider, such as Facebook, Google, and CDNs, may deploy an edge cluster including multiple servers (also called Points of Presence (PoPs)) that carry sufficient resources to store popular contents to allow users and Internet Service Providers (ISPs) to access these contents without reaching their data centers [27, 40]. Fig. 1 illustrates an infrastructure of edge content caching. Edge servers are deployed by network applications in different locations that are closer to users than the cloud, thus the latency could be reduced significantly due to content sharing.

As shown in Fig. 2 (a), in the classic edge caching service, when an edge server receives a user's data request, it searches the local storage of the edge server or edge cluster. If there is no hit, the edge server will forward the request to the cloud. Fig. 2 (b) shows a method called edge content sharing, which further reduces the

access latency: if the edge server finds no hit in local storage, it expands the search to edge servers and clusters in other locations and could find another edge server that holds the requested content. It then forwards the request to that edge server if it is closer to the user than the cloud. Edge content sharing is not a new idea and has been discussed before [8]. However, it is not widely used in practical networks [27, 31, 40]. The main challenge is that searching other locations requires either huge message costs to broadcast the request to other servers [8] or a gigantic directory storing the content-to-server mappings. Our experiments show that even with compact data structures such as counting Bloom filters (CBFs) [8], the DRAM cost per server is as high as 12GB to extend the search for other locations, including 1 million users based on the real distribution of contents generated by users [4].

We identify the major problem that is studied in this work: **with growing deployments of edge servers, the capacity of content storage in edge clusters significantly increases, which provides potentially much higher hit rates of edge content requests. However, the large amount of content also causes difficulty in indexing, which makes it hard to find the correct edge locations for content requests.**

This work presents a novel solution for scalable edge content access. We argue that a large amount of user-associated content provides a unique opportunity to enable edge content sharing with efficient and scalable directories. Our solution is based on two facts. 1) It has been reported that most accesses of contents associated with a specific user u happen in the physical regions close to the user's active areas [31, 39], i.e., the physical locations u frequently log in. For example, the users that frequently visit u's contents could be u's family members, friends, classmates, and colleagues. For video-sharing applications, users in the same area (city, province/state, or country) are more likely to visit their uploaded videos compared to others. Hence caching u's contents on the edge servers physically close to u's active location can increase hit rates if content sharing is allowed because most requests to u's contents also happen in these areas. 2) We allow all u's contents caching on the same edge server s (or edge cluster). A large amount of user-associated data provides a new yet not fully investigated opportunity for cache sharing within a large network application: Maintaining these user-to-server mappings $< u, s >$ is more efficient than maintaining content-to-server mappings. Moreover, our solution is easy to achieve consistency because the frequency of user location changes is much lower than that of content changes.

We develop an **Edge Location Service (ELS)**, called EdgeCut, that returns the responsible edge server of caching a user u's contents. We design a compact and dynamic data structure called Ludo Locator to support queries of edge servers and simultaneously distinguish mobile and non-mobile users. With Ludo Locator, for each content request, an edge server can quickly get the IP of the edge server that stores the content. We have implemented EdgeCut in a real network environment running in CloudLab [1], a public geo-distributed cloud. The extensive experiment results show that EdgeCut can reduce content access latency by 60% or more and reduce traffic to the cloud by up to 50%, compared to existing solutions. The memory cost is less than 48MB for extending the cache sharing scope of 10 million mobile users, less than 1% of the memory cost of CBFs. Note that practical networks also include

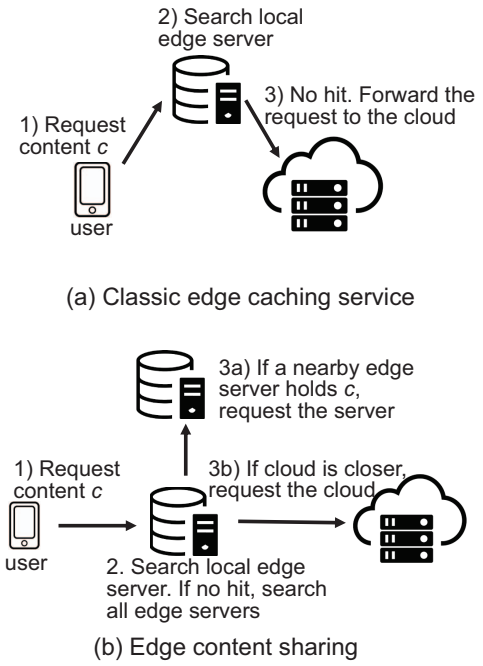


Figure 2: Edge content sharing can reduce access latency.

non-user-associated data, and they can be handled using state-of-the-art caching methods. Hence *this work is to complement existing methods rather than replacing them.*

There have been some works [29][8][38][33][11][37][35][36] to solve the problem of accessing user data among geo-distributed servers and edge servers. However, they either focus on designing better data replication method [38][33][11] or using routing schemes to find data [37][35]. They treat each content as an individual one and do not deal with user-associated data. Hence they cannot be used as an ELS. **To our knowledge, there is no solution in the literature for memory-efficient ELS that can be hosted on heterogeneous edge servers.**

Our contributions are summarized as follows:

1). We design a new protocol called EdgeCut to provide an ELS for user-associated contents. EdgeCut is the first ELS solution that is efficient to run on every edge server.

2). We design Ludo Locator, an efficient key-value store to maintain the mappings of users to their responsible edge servers while filtering non-mobiles users. It supports dynamic changes.

3). We implement a prototype of EdgeCut in CloudLab [1], a public geo-distributed cloud, for evaluation. We use different traffic datasets collected in the real world to show EdgeCut's performance on a large scale. The results show that EdgeCut can reduce content access latency by up to 50% compared to a recent solution that uses 3x storage resources. EdgeCut also reduces the cloud access rate by 30% to 50%.

The paper is structured as follows. We present the system model and problem statement in Section 2. The full design of the EdgeCut is presented in Section 3. We show the evaluation results in Section 4. Section 5 presents the related work, and Section 6 concludes the work.

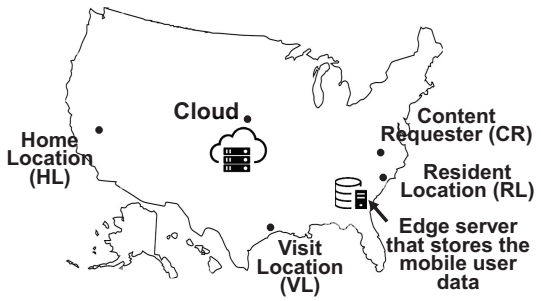


Figure 3: Several locations associated with one mobile user.

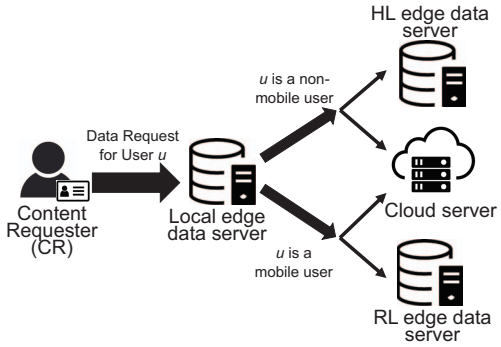


Figure 4: The workflow of EdgeCut.

2 MODELS AND PROBLEM STATEMENT

2.1 System model

We consider the scenario in which application users carrying end devices generate, share, and request contents under the edge computing infrastructure [7]. The contents discussed in this work are all user-associated and each content ID is queried along with its user ID *UID*. Fig. 1 shows the system model, including five main components.

1) **Network application.** The network application runs services that provide contents to end users, such as social networks, video streaming, and file sharing. It can be a single large application such as Twitter and Facebook, or a multi-tenant content provider such as AWS. The network application maintains the cloud and edge clusters, which are connected by its private WAN.

2) **Locations**. Each location is an area including multiple users and one edge server cluster. The size of each location is determined by the network application and is typically a city or county.

3) **End users**. Users carrying devices request contents from edge or cloud servers. Each user has a global-unique UID, assigned when the user registers, and the location of registration is called its home location (HL), as shown in Fig. 3. An HL covers an area where the content provider deploys at least one edge server or edge cluster. Hence users within a certain geographic area share the same HL. The user's HL would be encoded and put into UID as a prefix by the servers when they register. Since users do not always stay in their HLs, a user could be active in another geographic location for a long time until present and such location is called its *resident location* (RL). Such users are called the *mobile users*. A mobile user's active location is the RL and a non-mobile user's active location is the HL. Hence the contents of a mobile user are cached at an edge server in his RL and the contents of a non-mobile user are cached in his HL. When a user log-on in a new location, this location is marked as its visit location (VL). If he stays in the VL for a sufficiently long time, e.g., more than one month, the VL becomes his new RL. A user u can also be a content requester (CR) to request the content associated with another user v. A content request will first be sent to an edge server to check if the request can be satisfied by the cache. If not, the edge server will use EdgeCut to search other edge servers or cluster that is responsible for caching v's contents.

4) **Edge servers**. A *server cluster* including at least one edge server is deployed in each location by the network application to provide caching service to users in that location [7]. For each region, there is one control program that is responsible for managing the cached content of its edge servers, which can co-locate with an edge server.

5) **Cloud servers**. Cloud servers are the archive of user contents. A content request can always be satisfied by the cloud, but the cloud is remote to most users.

2.2 Problem statement

We target the problem of efficiently accessing cached user-associated contents from edge servers via edge content sharing. Each content request will be first sent to a local edge server of the content requester (CR). For user-associated content, the server can get the user ID (UID) with the associated content. To serve the request, the main challenge is to obtain the IP address of the nearby edge server that is responsible for caching the contents of that user, called the server IP (SIP) of that UID. The edge server could be either located in the HL for non-mobile users or RL for mobile users. Hence every edge server should be able to resolve the above information for an arbitrary UID. Note that if the request is forwarded to SIP and the content is cached within the edge cluster containing SIP, we consider the server with SIP can search its edge cluster and return the content to the requester. We do not discuss how contents are stored and searched *within* an edge cluster. EdgeCut relies on existing protocols to handle these tasks, such as the consistent hashing method used in Akamai edge clusters [21].

3 DESIGN OF EDGECUT

3.1 System Overview

EdgeCut includes one control program running in each edge cluster and one ELS program on every edge server. Fig. 4 shows the overview of the EdgeCut protocol. The workflow for a CR to request the content associated with user u is as follows.

Step 1. The CR sends the request for the queried content to his edge data server. The server gets the *UID* of u along with the content ID and makes a query to the ELS program on its local memory. The result is a tuple $< f, SIP >$ where f is a k-bit fingerprint of a *UID* computed by a hash function, and *SIP* is an IP address.

Step 2. The local edge server uses the fingerprint f in the query result of Step 1 to determine if u is a mobile user.

Step 3. Case 1: If the fingerprint matches the hash of *UID*, u is a mobile user. Then *SIP* represents the preferred server (edge

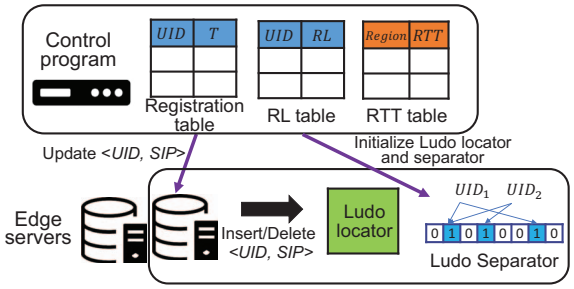


Figure 5: System overview.

server or cloud) that stores the contents of user u. Whether an edge server is preferred to the cloud can be defined by the provider. For example, the edge server is preferred if the RTT to it is less than α times the RTT to the cloud where α is a parameter such as 0.8. Note that *SIP* can be a virtual IP of the cloud that will be resolved later by its load balancer. The local edge server then forwards the content request to *SIP*, and that storage server will return the requested content to the CR. Case 2: If the fingerprint does not match, u is a non-mobile user. In that case, the *SIP* field will not be used. The edge server will forward the request to the HL edge server of the user or cloud, whichever is preferred. The HL of the user can be retrieved from its *UID*.

The key ideas of these design choices are: 1) managing content cache on the granularity of the user level rather than the content level, to reduce complexity; 2) separating the protocols for mobile and non-mobile users; 3) locating mobile user's RL using a memory-efficient lookup table. These design choices are based on the fact that the number of users is much smaller than the number of contents [4], mobile users can be easily located based on their UID, and RL lookups need to be performed in an efficient way.

3.2 Overview of data structures in EdgeCut

To complete the workflow discussed above, each edge server maintains two memory-compact and dynamic data structures developed by us, called Ludo Locator and Ludo Separator, as shown in Fig. 5. Ludo Locator maintains the $< UID, SIP >$ pairs for all mobile users and provides the result $< f, SIP >$ of querying a UID. The design of Ludo Locator is motivated by a recent key-value lookup engine called Ludo hashing [28]. We use Ludo hashing because it supports key-value lookups with memory efficiency and fast speed. The memory benefits of Ludo come from the feature that it does not store the keys. In our context, the keys are UIDs that could be very long compared to the SIP. Hence, avoiding storing the UIDs in the search engine is an ideal design choice. However, Ludo hashing has a limitation that prevents it from being directly used in EdgeCut: It returns an arbitrary result for any UID that is not included in it – in the case of EdgeCut, querying a non-mobile user will get a wrong *SIP*. Hence, we develop the Ludo Locator that can identify non-mobile users. To further decrease the miss-classification of non-mobile users, we develop the Ludo Separator.

Both Ludo Locator and Ludo Separator are constructed in the control program, and the control program sends them to edge servers via a software interface. For an edge cluster, one control program is sufficient to manage the data structures on all servers. Hence, such construction-query separation significantly reduces the average resource overhead on each server. The control program maintains three tables. 1) The registration table stores all users that use this region as HL, RL, or VL and the time they have been staying in this region. 2) The RL table stores the $UIDs$ of all mobile users and their RLs and the *SIP* of the edge server that stores the user contents. 3) The RTT table stores the round trip time (RTT) to every other region of the service provider and each region is identified by the IP address of its control program. All control programs will periodically synchronize the RL information of the mobile users while measuring the RTTs using these messages.

At the initialization of an edge server, the control program will compute its *Ludo Locator* and *Ludo Separator*, which are used to provide lookup services to tell 1) whether the user of the requested content is mobile, 2) if yes, what is the *SIP* of its RL, 3) whether the edge server should forward the request to that *SIP* instead of forwarding it to the cloud. The edge server directly receives these two data structures from the control program. After initialization, any changes of the $< UID, SIP >$ pairs can be updated by each edge server *locally*. The control program only needs to tell each server to insert or delete certain pairs. Such a design reduces the communication cost between the control program and servers.

When a CR wants to access content that is associated with user u, the content request is first forwarded to the edge data server in the region where the CR resides. On the edge server, Ludo Locator can provide a fast lookup service for the *SIP* of u, which contains the IP address of the edge server or cloud server, whichever is closer to the CR. The lookup result allows the edge server to forward the content request to that server. Each mobile user's location pair is stored in this table. The details of the Ludo Locator will be presented in the following part.

The locations of mobile users change dynamically, so the control program of each location maintains a *registration table* that stores the starting time of each mobile user arriving at that location. The control program maintains and updates the location status of each mobile user – determines whether this location is the user's VL or RL – according to the arrival time. The user-to-RL mappings of all mobile users are stored in the *RL table*. When there is a change in a user's RL, this information should be forwarded to control programs of all edge locations, which triggers the updates of their RL tables. When there is a change to the RL table, the control program can insert or delete user-location pairs to/from Ludo Locator. In addition, each control program also maintains the RTT estimations for communicating with all other edge control programs, and these estimations can be used to determine the network distances to other locations and whether other locations should be chosen favorably over the cloud if a request should be forwarded. The RTT estimations are computed using the moving average method like that in the TCP protocol and stored in the *RTT table*.

Ludo Separator is a memory-efficient (2,1)-Cuckoo filter [12] running on each edge server to further reduce the false positives caused by the Ludo Locator while distinguishing mobile users from non-mobile users, assisting to provides a more accurate ELS for users. The detailed design of the Ludo Separator is presented in Sec. 3.5.

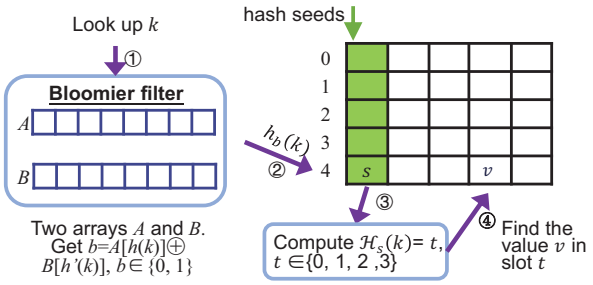


Figure 6: Ludo hashing

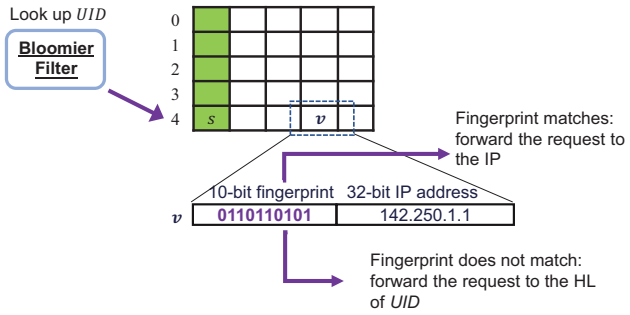


Figure 7: The workflow of Ludo Locator

Another advantage of EdgeCut is the ability to remove non-mobile users from the whole user set without storing their keys. However, the number of users is so large that just using a fingerprint of a certain length of bits will result in a large number of false positives. The core structure of the Ludo separator is the (2,1)-Cuckoo filter, which can make the number of false positives decrease exponentially with a linear increase in memory usage. The reduction of the false positive rate provides accurate location information of mobile user contents, thus reducing invalid data access times and reducing the burden on the communication network.

3.3 Ludo Locator

One challenge for providing a memory-efficient ELS on each edge server is finding a space-efficient data structure to store UID-to-SIP mappings and support fast lookup.

We briefly introduce Ludo hashing before presenting our design. The proposed Ludo Locator is motivated by a recently developed data structure called Ludo hashing [28], which Ludo Hashing is a key-value lookup engine with small memory cost, > 50% smaller compared with several known compact data structures such as Cuckoo hashing and Bloomier filters [28] while supporting a fast key-value pair updates.

Fig. 6 shows the structure of Ludo hashing, which is a tuple $\langle O, T, h_0, h_1, \mathcal{H}\rangle$, where O is an Othello lookup structure [41] that maintains two bitmaps A and B and returns 1-bit value to indicate whether the value v_k of a key k is stored in the bucket $h_0(k)$ or $h_1(k)$ of T. Here, T is a (2,4)-cuckoo hash table [25] that stores the values of keys. $\mathcal{H}$ is a universal hash function family. The index t in the bucket is determined by comping $t = \mathcal{H}_s(k)$ $(t \in 0, 1, 2, 3)$ where s is the seed stored in each bucket. Ludo hashing does not need to store keys. For each key k, Ludo hashing first queries the Othello structure to get whether the value is stored in the bucket $h_0(k)$ or $h_1(k)$. Then Ludo hashing will extract the seed s in the target bucket and compute the index $t = \mathcal{H}_s(k)$. Finally, the value stored in the slot t will be returned as the value v_k. Ludo hashing is memory-efficient, which only costs $(3.76 + 1.05l)$ bits per key-value item for l−bits values, providing > 50% space reduction compared to existing solutions. Ludo hashing is fast in lookups and updates. The time of item lookup and deletion is $O(1)$ and the time of item insertion is amortized $O(1)$ [28].

For EdgeCut, the 'key' in our context is the UID of a mobile user, and the 'value' is the IP address of the edge server or cloud, whichever is preferred. The limitation of directly applying Ludo hashing in EdgeCut is that the Ludo hashing cannot tell whether the given UID is for a mobile user or a non-mobile user. Since non-mobile users are not stored in the table, by querying a UID of a non-mobile user, Ludo hashing will return an arbitrary SIP. The incorrect SIP for a non-mobile user will bring extra response time and traffic. One intuitive solution to solve this issue is to place a Bloom filter [10] that stores the user IDs of mobile users before querying the Ludo hashing to filter non-mobile users. However, Bloom filters introduce extra memory costs and memory accesses. Instead, we present a solution that still uses three memory accesses per lookup in the Ludo hashing.

Ludo Locator is designed to identify non-mobile users while using less memory and fewer memory accesses compared to putting a Bloom filter. Instead of storing a value, Ludo Locator stores a tuple in each table slot. The tuple includes two fields: (1) a fingerprint field, also called user check code, which is a l_c−bits value, and (2) the SIP. The fingerprint stores the partial hash value $\mathcal{H}_s(UID)$, where $\mathcal{H}$ is the uniform hash function used in calculating the slot index. Hence there is no extra step of hash computation. Fig. 7 shows the slot format in Ludo Locator. To query a UID, Ludo Locator finds the corresponding slot and gets the tuple. Then it compares the fingerprint with $\mathcal{H}_s(UID)$. If they match, Ludo Locator concludes that the user is a mobile user and the SIP can be used for forwarding. Otherwise, the user is regarded as a non-mobile user. The content request can be forwarded to the edge server of the user's HL. Recall that the HL ID is stored along with every UID. Hence each edge server also stores a small hash table for HL ID to HL IP mappings. Such a table can be implemented with a standard Cuckoo hash table [25].

We should point out that due to hash collisions, different users may have the same fingerprint. If a non-mobile user is mapped to a slot and the fingerprint matches its UID, we call this a *false positive* (FP). The FP rate of an arbitrary non-mobile user for Ludo Locator is 2^{l_c} where l_c is the length of each fingerprint. We set the default l_c to be 10 bits in our design, leading to an FP rate as low as 1/1024, which can be further reduced by the design presented in Sec. 3.5. Suppose that a non-mobile user u is incorrectly identified as a mobile user M; Ludo Locator will return the SIP of M. In this case, the target edge server unlikely stores the contents of u and will forward the request to the cloud. Note that for all mobile users, all results provided by Ludo Locator are correct.

Lookup overhead. Each query lookup of the Ludo Locator requires only four hash computations, two for the Othello lookup,

one for calculating the bucket index h_b, and one for the slot index $\mathcal{H}$ inside the bucket. The fingerprint computation reuses the partial result of $\mathcal{H}$. If the seed for one bucket overflows, additional 1 or 2 hash computations are needed to get the result from the fallback table, but the probability of bucket overflow is low (less than one out of a million, according to our experiments).

Insertion and deletion overhead. Ludo Locator updates dynamic changes of mobile users' RL information. Once a non-mobile user becomes a mobile user, or a new mobile user joins the system, the user's $< UID, SIP >$ pair is inserted into the table. The insertion time consists of 1) an update to the Othello structure in $(O(1))$ time; 2) an update in the (2,4)-Cuckoo table contained in the Ludo Locator in $O((m/m-n)^{O(\log(\log(m/m-n)))})$, where m is the number of buckets in the Cuckoo table and n is the number of mobile users; 3) the edge control program broadcasts the insertion update information to the other regions. A similar but simpler process is conducted for a deletion operation.

Selection of SIP in Ludo Locator. The control program constructs Ludo Locator for edge servers from the RL table that stores the full mappings of UIDs to their SIPs in RL. It needs to decide which IP address should be put into the *SIP* field for each mobile user with *UID*, specifically, the SIP of RL or the IP address of the cloud. Both are correct locations that store the contents of *UID*. Whether an edge server is preferred to the cloud can be defined by the network application. A possible condition can be the RTT to the edge server is less than α times the RTT to the cloud, where α is a constant such as 0.8.

3.4 Update and consistency of Ludo Locator

Mobile users may also move to other locations and their RLs may change. How to handle the dynamics to maintain correct lookup tables is a challenge.

The control program tracks the mobile users in its location using the registration table. When a mobile user visits the location, the registration table records its visited time associated with its UID. The time will be used for tracking if the user's VL should be changed to its RL. When a non-mobile user changes to a mobile user or a user's VL changes to its RL, an update is necessary to edge servers to reflect the correct mobile user to SIP mapping.

The edge data server only needs to store one table, maintaining the mobile user ID and its associated RL edge data server IP. However, if the network latency required for the edge control server in this region to communicate with the cloud server is lower, then the value saved in the RL field is the IP address of the cloud server.

RL insertion. When a user u moves into a new location r, the location is marked as its VL, and the location's control program records it in the registration table. The contents generated by u are cached by an edge server s in r, and s also forwards the contents to u's corresponding server s' in its RL or HL by looking up Ludo Locator. When the control program in r finds that the time u has been in this area exceeds a predefined threshold T_{in}, it determines that it becomes the RL for u. The edge server s in r will be the new corresponding server of u and notify the current RL edge server s' to stop caching more u's contents and migrate the existing cached contents of u to s. The IP address of s will be the SIP of u. The control program notifies all edge servers in r to perform an insert operation of $< UID_u, SIP_s >$ in their Ludo Locator, where SIP_s is the IP of s. The control program also synchronizes this information with other locations. The control program of every other region will tell its edge servers to insert $< UID_u, SIP >$, where SIP is the IP address of s or the cloud, whichever is preferred.

Note there is no difference in the RL insertion process for the case of u moving from its HL to an RL or the case of moving from a previous RL to a new RL. If a previous mapping of u exists in Ludo Locator, it will be replaced by the new one. If no mapping of u exists, a new one will be inserted. In the end, the resulting Ludo Locator is the same in the two cases.

RL deletion. RL deletion happens in only one case: a mobile user becoming non-mobile. When a mobile user u returns to its HL r, an edge server s of r starts to store new contents generated by u. When u stays in its HL for a longer time than the threshold T_d, the control program of r will notify the current RL edge server to stop caching more u's contents and migrate the existing cached contents of u to s. The mapping of u in the control program of every location will be deleted. EdgeCut supports quick deletion operations by simply deleting an entry from the control program and removing u's fingerprint in Ludo Locator.

Consistency. Maintaining consistency ensures the lookup results of the Ludo Locator reflect the actual edge servers that store the requested contents. If a mobile user u moves from an RL to a VL and stays long enough, the VL becomes the new RL, and the previous RL should retire. EdgeCut will request the RL server s' to stop caching more contents of u and migrate the existing contents to the VL edge server s. The migration should be complete before the changes to Ludo Locator take effect. During this period, any changes about u's contents that are sent to s' will be forwarded to s.

If user data that has been copied to the VL server is written, the RL edge control server can act as an anchor node to inform the VL edge data server to modify the corresponding value. The write operation in these two servers can be asynchronous because any concurrent writes to the same content are subject to the last writes to the RL edge data server until the RL value is changed. Copying before RL value modification in the Ludo table ensures the consistency of data before and after the user moves. The same is true for mobile user deletions and new insertions. For the consistency of the Ludo Locator table, an edge control server sends the update information to other edge control servers once there is an RL insertion, update, and deletion.

Size setting of Ludo Locator. The load factor is defined as the number of key-value mappings over the total number of slots in a data structure such as Cuckoo hashing, Ludo hashing, and Ludo Locator. It has been shown that insertions never fail when the load factor of Ludo is <97% [28]. It proves in theory that the insertion would always be successful asymptotically when the load factor is <98.03% [15]. In our implementation, as long as the load factor of a Ludo Locator is below 97%, reconstruction will not be triggered. Otherwise, the control program will start reconstructing the Ludo Locator by increasing its size by 10% for all edge servers in its location.

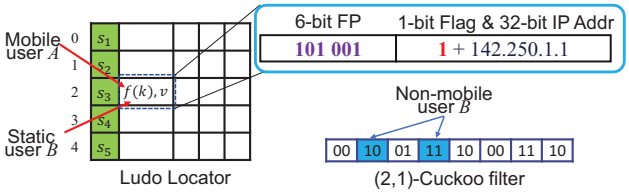


Figure 8: Construction of the Ludo Separator.

3.5 Ludo Separator

When the number of non-mobile users is large, Ludo Locator's fingerprint-based approach still produces false positives that categorize some of them as mobile users (shown in Section 4). However, a unique feature that can be utilized by EdgeCut is that, once a false positive is detected by an edge server, it can be recorded and later queries of the same non-mobile user will not cause a false positive again.

Towards this goal, we design Ludo Separator, a secondary data structure based on a (2,1)-Cuckoo filter [12], i.e., a Cuckoo hash table that uses two alternate buckets for each key and each bucket stores the fingerprint of the key mapped to this bucket. To query a UID, Ludo Separator fetches two buckets and checks if one of the fingerprints matches the fingerprint of the UID. If there is a match, Ludo Separator returns a positive result, otherwise it returns a negative result. In addition, the SIP field in Ludo Locator includes an extra flag bit to indicate a collision in this slot. Suppose that one slot in the Ludo Locator stores the fingerprint and the SIP corresponding to a mobile user a, and a non-mobile user b is also mapped to this slot, and they happen to have the same fingerprint. Then the flag bit is set to 1 to indicate that there is a potential false positive so the edge server will further lookup b in Ludo Separator.

The construction of the Ludo Separator is as shown in Fig. 8. When a local edge server s processes the request of a content of a non-mobile user u but encounters a false positive, it will forward the request to an edge server s' that does not store u's contents. s' will report this false positive to s, so s should insert the element u into Ludo Separator and set the collision flag bit of the slot in the Ludo Locator to 1. In Ludo Separator, each bucket contains a 2-bit fingerprint. If there is again a fingerprint match of u in Ludo Separator, EdgeCut can conclude that u is a false positive and thus is a non-mobile user. Otherwise, u will be forwarded to s' because it is not a false positive. After every insertion, the edge server broadcasts its Ludo Separator to all edge servers within the same cluster, to make all Ludo Separators consistent.

The (2,1)-Cuckoo filter can be used during the construction. When a CR wants to request the user's data, the edge data server first checks whether the flag bit is "1" after finding the SIP corresponding to the user ID. If the flag bit is "1", then the user ID needs to be checked in the Cuckoo filter again using the new identifier.

In summary, the fingerprint of the Ludo Locator removes most non-mobile users, and the second filter – Ludo Separator removes collisional mobile users. Thus, the false positive rate will be greatly reduced, and the separation of mobile users and non-mobile users will be completed more thoroughly. The evaluation results show the false positive rate using Ludo Locator plus Ludo Separator is more than 80% lower compared to using Ludo Locator only with more memory cost.

We conduct the following analysis on the false positive rate by using both Ludo Locator and Ludo Separator.

Suppose there are n mobile users and m non-mobile users and we assume the ratio of them is β, which means $m = \beta n$. As for the locator-only approach, the length of the fingerprint is l_1. For the Ludo Separator approach, the length of a fingerprint in the Ludo Locator is l_2 and a fingerprint in (2,1)-Cuckoo filter is l_3 bits. For the locator-only approach, the memory cost is: $n \cdot (32 + l_1)$ and its false positives only come from non-mobile users is: $m \cdot \frac{1}{2^{l_1}}$.

As for the Ludo Separator, its memory cost consists of the Ludo Locator and the additional (2,1)-Cuckoo filter, The length of the Cuckoo filter follows the setting in [12]. Thus its memory cost can be expressed as: $(l_2 + 1 + 32) \cdot n + \frac{l_3}{2^{l_2-1}} \cdot n$. The false positives of Ludo Separator come from the mobile users whose UID pass the Ludo Separator and are identified as non-mobile users: $n \cdot \frac{1}{2^{l_2}} \cdot \frac{1}{2^{l_3}} + o(n \cdot (\frac{1}{2^{l_2}})^2)$, where $o(n \cdot (\frac{1}{2^{l_2}})^2)$ represents the probability of cases that two or more non-mobile users are mapped to the same slot with one mobile user and their fingerprints are all the same. Since the value is very small, we ignore the calculation here.

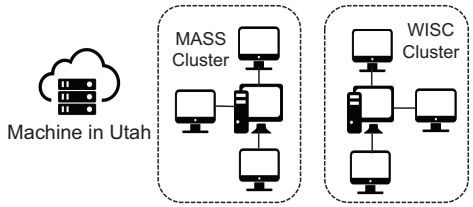
Suppose the memory cost of these two schemes same, then the l_1, l_2, and l_3 will follow this equation: $l_1 = l_2 + \frac{l_3}{2^{l_2-1}} + 1$. Thus, when β is greater than $2^{1+\frac{l_3}{2^{l_2-1}}-l_3}$, the Ludo separator will outperform the fingerprint-based approach in the number of false positives.

3.6 Scalability.

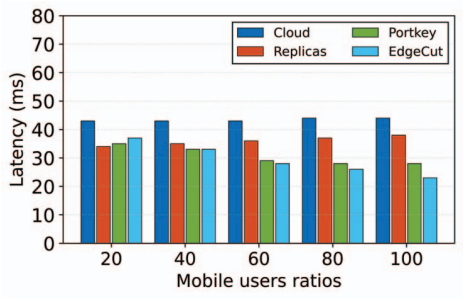
When the number of mobile users reaches the threshold of the load factor (e.g., 0.85) of the registration table or there is a deadlock encountered when inserting a new UID, the registration table maintained in the control program (shown in Fig. 5) would be enlarged. Then, all $< UID, SIP >$ pairs will be inserted into the new empty table again. However, new $< UID, SIP >$ pair insertion would be halted during registration table resizing. Even if the mobile users' registration is not a time-sensitive task (e.g., transaction system), the time and CPU overhead for migrating all current $< UID, SIP >$ pairs into an empty table is large. To reduce the halted time for the registration table when resizing, we leverage the extendible hashing scheme to reduce the number of migrated pairs. In extendible hashing, there is only one registration table with the fixed size first; when the number of $< UID, SIP >$ pairs in it reaches a threshold, there is another table would be created with the same size, and all the UID ending with the "0" would stay in the original table. The left pairs whose UID ends with "1" will move to the new split one. When both of them become full, two new registration tables would be created to have the pairs whose UID ends with "10" and "11", and the UID ending with "01" and "00" will stay in the original tables and keep going. Then, there is always the power of two registration tables kept in the control program, and only half of $< UID, SIP >$ pairs would be moved to the new registration tables.

3.7 Data privacy.

From a trustworthy edge computing perspective [32], the protection of user-associated data is of paramount importance due to the



(a) The network topology consists of 9 machines from MASS, WISC and Utah clusters in the CloudLab.



(b) Average latency of accessing user-associated content.

Figure 9: Results from EdgeCut prototype implementation.

sensitive and private nature of the information involved. Safeguarding this data is crucial to ensure individuals maintain control over their personal information. It is important to note that in this particular context, the focus is on the content and data published by the user. The EdgeCut framework enables CR to query user-associated content based on the user's unique identifier (*UID*). Throughout the entire data query process, the user's identity, who publishes the content, remains undisclosed to the CR system. This privacy assurance is achieved through the implementation of a control program running in the application layer. By operating at this layer, the control program guarantees that the user's information is not exposed to the CR. Additionally, content owners retain the authority to establish access permissions for other users attempting to access their content through the use of metadata. This feature empowers content owners to set appropriate levels of authorization, granting or denying access to specific individuals or groups as they see fit. By emphasizing privacy protection, incorporating control mechanisms, and empowering content owners with access authority, the framework aligns with trustworthy edge computing principles, providing users greater control and security over their personal data.

4 EVALUATION

We evaluate the performance of our EdgeCut against a few state-of-the-art methods that can be utilized for edge content access, by both the prototype implementation and simulations with real network datasets.

4.1 Application workloads

The user-associated content is indexed with the *UID*. We use the YCSB workload [26] to generate 10M distinct keys as *UID* for all users, which are spread in the whole network uniformly. In the latency measurement, we randomly select 10K users as CRs. For each CR's workload, we generate 1K *UID* as their content access workload in the YCSB suite with the distributions of uniform and Zipfian-0.9. In the following evaluation, we will change the mobile users ratio in each CR's workload to get the average latencies, which will be treated as the lookup latency.

4.2 Methods to compare with

We compare EdgeCut with the following three approaches in the evaluation:

1). Direct cloud access (denoted as "Cloud"): A naïve approach that directly queries the cloud to get every content.

2). Proactive replication: In the proactive replication scheme [13][14], edge servers provide content access based on the "cell structures", and proactively replicate data to the different number of edge servers in the adjacent cells. For our evaluation, we replicate user contents from each edge server into 2 adjacent hexagonal cells. When a CR requests a mobile user's data, the local edge server will access one of these three servers.

3). Portkey [24]: This method is an adaptive placement for a distributed KV store in the case of dynamic edge networks. In Portkey, all KV pairs are grouped with the hash function. The main strategy is to place different KV pair groups in the edge server where it can achieve low latency. The optimal placement for each KV pair group comes from the analysis of the workload and the applied greedy algorithm. In our implementation, the "optimal" location for each KV pair group is set on the edge server where users request them most.

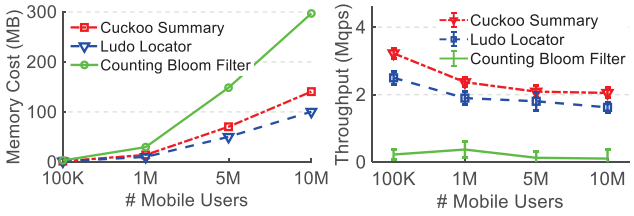
4.3 Implementation and prototype evaluation

We implement the EdgeCut protocol as a prototype system running in CloudLab [1], a public geo-distributed cloud. The prototype consists of 9 machines from 3 different clusters located in Massachusetts (MA), Wisconsin (WI), and Utah (UT). As shown in Fig. 9a, four physical servers are used in each of MA and WI as an edge cluster: three computers are used as edge servers connected to one physical machine as the control program. In addition, we also use a machine in UT as the cloud. We use Redis [5] to store user IDs and a random string (1KB) as value, the ZeroMQ [6] will be leveraged to complete communication between machines.

In EdgeCut, All UID-to-SIP mappings are stored in Ludo Locator, running on each edge server in MA and WI. Each edge server performs queries of contents from different mobile users 10 thousand times and sends the content requests to the targeted servers indicated by the SIPs. Each successful content request starts with the requester sending the request and finishes at the requester receives the requested content. Meanwhile, we record the response latency for these content requests and calculate the average response latency and throughput of satisfying content requests on each edge server.

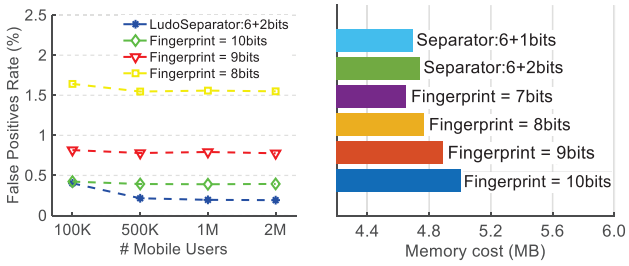
Num. of users	100K	1M	5M	10M
CBF (content-level) [8, 21]	1190	11870	59360	118710
EdgeCut (user-level) (this work)	0.5	5.1	25.1	50.2

Table 1: Memory cost (in MB) for content-level and user-level indexing structures.



(a) Memory cost. **(b) Lookup throughput.**

Figure 10: Performance comparison for Ludo Locator, Cuckoo summary and counting Bloom filters.



(a) The false positives rate. **(b) The memory cost.**

Figure 11: Evaluation of Ludo Separator against locator-only approaches.

In the proactive replication approach, we store three copies of each content in another 2 adjacent edge servers. Also, CR can reach any of these three replicas (including HL) to access queried data.

In Portkey, we cached the mobile user's contents in the edge server based on the analysis of CRs' workloads. The location of the edge server is closer to the CR that queries this mobile user's contents most frequently. Note that all UIDs are hashed into different groups to be placed on different servers, so the cached edge server may not be the closest location for CRs to access the user-associated content.

We distribute all users in the above two clusters and generate queried workloads for randomly selected CR from the YCSB workload with the Zipfian distribution with a constant 0.9 [17, 19, 20]. For mobile users, their RL and HL are uniformly distributed in two distinct clusters. Fig. 9b shows the latency of completing content requests on each edge server. No error bars are drawn in the figures because the variations are too small to see. EdgeCut can achieve 0.5x latency compared to direct cloud access when the mobile users ratio is more than 60. For mobile users, CR can get a closer SIP from the Cloud or RL cached server for the queried *UID* via the Ludo locator. For non-mobile users, HL or Cloud will be chosen to access data based on the round trip table. Portkey can choose an edge server for caching all user-associated data in a user group, but that may not be a closer choice for each queried user data.

4.4 Simulations with real network datasets

We use the network latency datasets [43] based on PlanetLab [3] and the Seattle topology [2] for evaluation. The PlanetLab dataset collects the pair-wise RTTs among 490 nodes over 18 time slices in the PlanetLab network. The Seattle dataset consists of pair-wise RTTs among 99 nodes in 688 time slices. We choose one node as the cloud and others as edge clusters. We use one workstation with two Intel E5-2660 v3 10-Vore CPUs at 2.60GHz, 160GB 2133MHz DDR4 memory, and 25MB LLC to run the simulations.

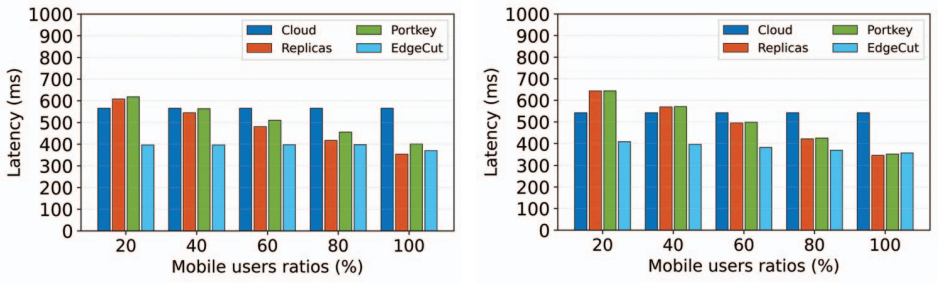
It is well known that contents have different popularity and both content popularity and user activities may be described as Zipfian distribution [17]. We generate two types of content request workloads in both uniform and Zipfian distributions to show that Edgecut can be widely applied in different scenarios based on YCSB workload [26], which is a common setting [20]. For the uniform distribution workload, each user's contents have the same probability of being accessed by others. The Zipfian distribution (constant 0.9) workload includes a small part of popular *UIDs*.

4.4.1 Evaluation of Ludo Locator. We implement Ludo Locator using C++ code and compare it with two data structures used for recent indexing solutions named Cuckoo Summary (CS) [37] and counting Bloom filters (CBF) [21]. In our implementation, each CBF is constructed by an edge cluster based on the mobile users it is responsible to [21]. CBFs will be synchronized among edge clusters.

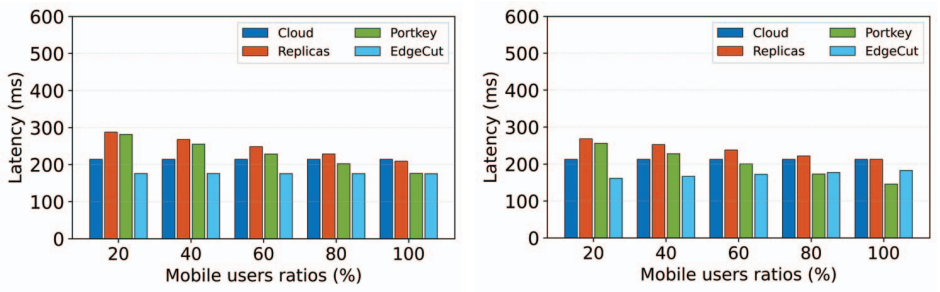
Memory cost. Fig. 10a shows the memory cost comparison of the three methods. Ludo Locator costs smaller memory than CS, in particular around 25% less for 10M mobile users, and significantly smaller memory than CBF, around 66% less. Note the memory cost of CBF in Fig 10a is for user-level indexing, while the original design of CBF is for content-level indexing. Table 1 shows the comparison of the Ludo Locator and content-level indexing using CBFs. The content-level indexing requires over 12GB for 1 million users based on the real distribution of the number of contents per user per year [4]. We find that content-level indexing requires 2000x more memory and hence is impossible to support cache sharing in modern edge computing. User-level content caching is a preferred solution.

Lookup throughput. We evaluate the lookup throughput in millions of queries per second (Mqps) of the data structures for the three indexing methods. We vary the number of mobile users from 100K to 10M. The test dataset contains 10K UIDs that could be repeatable. Each point value is the average time of 5 runs. Fig. 10b shows that CS achieves slightly faster throughput than Ludo Locator. However, Ludo Locator achieves around 2 Mqps, way more than sufficient to support queries on the edge server. Hence, the throughput Ludo Locator and CS do not make significant differences. Both of them are much faster than CBF.

4.4.2 Evaluation for Ludo Separator. In this section, we compare the approach of the Ludo Locator plus Ludo Separator (denoted



(a) Latency with uniform distribution in Seattle topology. (b) Latency with Zipfian distribution in Seattle topology.



(c) Latency with uniform distribution in PlanetLab topology. (d) Latency with Zipfian distribution in PlanetLab topology.

Figure 12: Comparison of content access latency.

as Ludo Separator in short) with the locator-only approach. The performance metrics include the number of false positives, false positive rate, and memory cost.

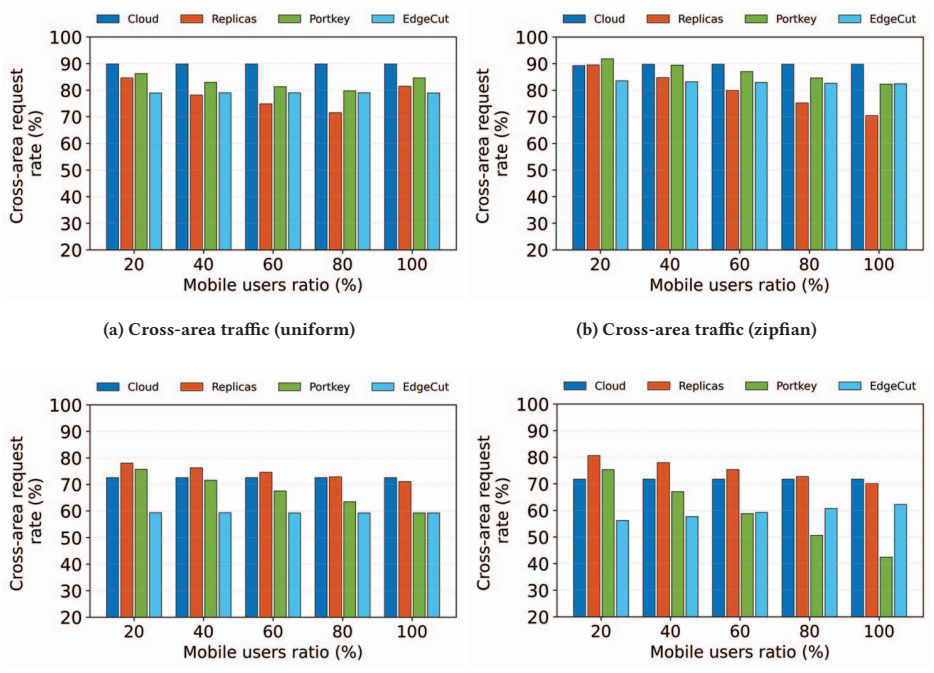
Ludo Separator is used in a two-layer data structure, including Ludo Locator and Ludo Separator, which is a (2,1)-Cuckoo filter. Ludo Locator can filter out most non-mobile users, and the Cuckoo filter can remove the mobile users that are false positives introduced by fingerprint collisions in the Ludo Locator. After this two-layer screening, the number of false positives can be reduced significantly. As for the fingerprint-based approach, it can use more bits for fingerprints in Ludo Locator to reduce the false positive rate.

We vary the number of mobile users from 100K to 2M and test with 4x non-mobile users. We assume EdgeCut uses 6-bit fingerprints in the Ludo Locator and 2-bit fingerprints in Ludo Separator for the solution that combines them. We compare the locator-only approach by varying the length of the fingerprint from 8 bits to 10 bits. As shown in Figs. 10a, Ludo Separator achieves the lowest false positives compared to all three versions of Ludo Locator-only approaches. We set 1M mobile users and compare the memory cost of all approaches. From Fig.11b, the memory costs of the Ludo Separator with (6+2) bits are smaller than that of the locator-only approach (8 bit). Combining the results of the Ludo Separator with (6+2) bits and the locator-only approach (8 bit) from all three figures in Fig. 11, we find that Ludo Separator costs lower memory while achieving a lower false positive rate with its two-layer design.

4.4.3 Evaluation for network latency. We test the latency of content accesses by comparing EdgeCut with Cloud and Proactive Replication. Lower latency infers the edge cache-sharing method with a higher fit rate. We implement EdgeCut with different mobile user ratios from 20% to 100%. In our benchmark, the CRs are randomly chosen on different servers. We also assign each CR with 10K queried users' IDs. We generate 1K *UID* as the queried workload based on the YCSB suite with the distributions of uniform and Zipfian-0.9.

Figs. 12a and 12b show the latency of the Seattle topology with uniform and Zipfian distributions, respectively. Figs. 12c and 12d show the latency of the PlanetLab topology with uniform and Zipfian distributions, respectively. In most cases, EdgeCut provides lower latency than the other three schemes, especially when the ratio of edge users is large. The reason is that the Ludo locator enables users to access the targeted mobile user's content from the closer *SIP* for each CR. In Portkey, even if the cached user data is placed in an edge server in a hashing group based on the analysis of CRs' workloads. However, each group contains many users' data; thus it is not optimal for all CRs to get data from a closer place. In the proactive replicas approach, user data is copied to the edge servers that are adjacent to HL and cannot reduce the latency for a CR that is far from them. Low latency means that the time taken for a user's request to reach the server and for the server to respond is minimal. The reduction of the latency provided by EdgeCut can enhance user satisfaction. Also, due to the different node geo-distributions in the Seattle and PlanetLab networks, the RTTs in these two datasets are different. Therefore, in the simulation results, the average latency of the PlanetLab network is lower than that of the Seattle network.

4.4.4 Cross-area traffic rate. Cross-area traffic refers to the data traffic that needs to be transmitted between different geographical



(a) Cross-area traffic (uniform)

(b) Cross-area traffic (zipfian)

(c) Cross-area traffic (uniform)

(d) Cross-area traffic (zipfian)

Figure 13: Cross-area traffic rate comparison.

areas or regions. In the context of the given statement, cross-area traffic specifically refers to content requests that are satisfied by edge servers located far away, resulting in longer latency. To evaluate the distribution of content requests among edge servers, the metric of cross-area traffic rate is used. This metric measures the proportion of content requests that are satisfied by distant edge servers or the cloud, leading to higher latency.

In our evaluation, cross-area traffic is defined as one with a latency longer than 120 ms by analyzing the network latency from CR to the targeted user-associated data. Fig. 13 shows the cross-area traffic rate of the three methods in both Seattle and PlanetLab topologies with different workloads, with different mobile users ratios. In all cases, EdgeCut can reduce the cross-area traffic rate by 5% to 30% compared to other methods. Hence, more requests are satisfied by EdgeCut on nearby servers.

4.4.5 Cloud access rate. Reducing cloud traffic load has emerged as a crucial objective in the realm of edge computing, prompting the pursuit of various solutions such as edge caching. EdgeCut can also be employed as a user-associated data caching approach to alleviate the burden on cloud resources by storing and delivering mobile users' data at the network edge.

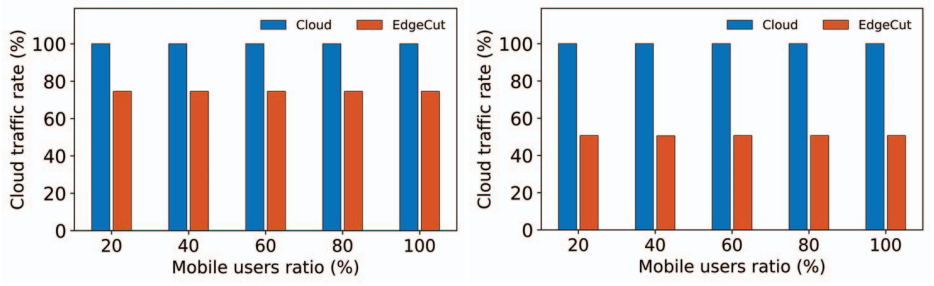
We show the cloud access rate of the three methods in Fig. 14. The cloud approach enables CR to request data from the cloud server each time. Thus, the cloud traffic reaches 100%, as shown in the red dot line. Again, with both uniform and Zipfian workloads, EdgeCut always achieves the lowest cloud access rate and reduces the metric by 20% to 50% compared to Proactive Replication, despite the latter using the 3x storage resource.

5 RELATED WORK

Reducing network latency and cloud bandwidth cost of content accesses is a crucial issue. Edge caching is one promising approach to achieve so [23].

Data Caching in Edge Computing. Xia *et al.* [34] models the collaborative edge data caching problem (CEDC) as a constrained optimization problem and proves it is NP-complete. They also propose an online algorithm, called CEDC-O, to solve this problem based on Lyapunov optimization. Zhang *et al.*[42] investigates delay-optimal cooperative edge caching in large-scale user-centric mobile networks. By proactively storing files at base stations (BSs) and utilizing cooperative caching, the study aims to reduce end-to-end delay and alleviate backhaul pressure and proposes a greedy content placement algorithm based on optimal bandwidth allocation. Gabry *et al.* [16] focuses on optimizing content placement in wireless edge caching to maximize energy efficiency in heterogeneous networks. By minimizing key metrics like expected backhaul rate and energy consumption through convex optimization, the study highlights a tradeoff between these factors. However, even if there are optimizations to solve the placement problem for data in edge computing, the Ludo locator proposed in this work can always enable the user to access them with a memory-efficient data structure.

Edge Location Services. The location service enables users to locate the specific edge servers that store the content they are requesting. In a recent study by Xie *et al.* [35], they introduced a highly efficient data indexing framework called HDS. This framework divides the forwarding of data access requests into two categories:



(a) Cloud access rate (Seattle, uniform) **(b) Cloud access rate (Seattle, Zipfian)**

Figure 14: Comparison of cloud access rate.

inter-region and intra-region. The index of data items is stored in the Cuckoo Summary, which is maintained in the regional data center. COIN [37] is another efficient routing method that focuses on finding cached content. It leverages virtual coordinates on P4 switches to optimize the routing process. However, both HDS and COIN rely on content-level indexing, which limits their ability to handle a large number of users and content effectively. In contrast, EdgeCut addresses the scalability issue by utilizing user-level indexing and caching properties. By leveraging these techniques, EdgeCut is able to provide a solution that overcomes the challenges posed by a large user base and a vast amount of content.

User Profile Replication. User profiles, which are essential user-associated data, have been extensively studied in the context of mobile networks [30]. In their work, the authors approach the user profile replication problem by formulating it as a flow network problem [18] in order to optimize the minimum communication latency to a replica. They also propose a replication mechanism that takes into account calling and user mobility patterns, resulting in faster location lookup.

Building upon this research, Shivakumar *et al.* [29] presents a novel replication mechanism for user profiles, leveraging known calling and user mobility patterns to enhance the speed of location lookup. However, in the context of the future era of software-defined networking and edge computing, there is a shift towards storing user profiles on local edge servers. Consequently, our primary focus is on efficiently determining user location while ensuring network scalability.

Nevertheless, in modern network applications, the indexing of user-associated data poses a challenging problem that existing works have yet to resolve.

Data replication provides several benefits, including load balancing, consistency maintenance, and fast access by duplicating data across different geo-distributed servers. Mansouri *et al.* [22] introduced a Dynamic Popularity Aware Replication Strategy (DPRS) that utilizes access history to prefetch popular data adaptively.

For mobile edge computing, Farris *et al.* [13] proposed a reactive/proactive data replication approach based on mobile user movement patterns. They formulate the data replication selection policy as an optimization problem, considering two metrics: Reactive Migration (RM) times and Numbers of Service Replicas (NSR). However, this approach often results in storing user data in more than three adjacent edge data servers, which is unnecessary in terms of memory consumption in the future. In Section 4, it is demonstrated that despite proactive replication storing more copies of data, EdgeCut still outperforms it in terms of access latency.

Edge data store. Edge data stores [24] play a pivotal role in edge computing by providing localized storage and efficient data access at the network edge. These data stores address the challenges incurred by latency and bandwidth limitations often encountered in edge computing environments.

An edge data store typically comprises a distributed storage system deployed in proximity to the devices and users generating the data. It enables the storage and retrieval of frequently accessed data or pertinent subsets of data that require rapid processing or delivery. We can apply different key-value store engines or data structures as the backend of the edge server. In EdgeCut, we leverage the Ludo locator and Ludo separator as two main structures. The indexing structure that is more memory efficient is the preferred feature for edge servers because the capacity is limited in edge servers.

Another benefit of edge stores is data privacy and security: With edge data stores, sensitive data can be stored and processed locally, reducing the risk of data breaches and ensuring compliance with privacy regulations. This localized approach enhances data privacy and security by minimizing data exposure to external networks. Our future work would focus more on how to construct and maintain trustworthy edge data stores.

6 CONCLUSION

This paper introduces EdgeCut, a scalable protocol that facilitates edge location services for accessing user-associated content. A key innovation of this work is Ludo Locator, an efficient key-value store that maintains UID-SIP mappings, filters non-mobile users, and supports dynamic changes. Additionally, we have developed a data structure named Ludo Separator, which significantly reduces the false positive rate of filtering while consuming minimal memory.

To evaluate the effectiveness of EdgeCut, we have implemented a prototype in CloudLab, a public geo-distributed cloud platform. Furthermore, we have conducted simulations using diverse large-scale datasets collected from real-world scenarios. The results demonstrate that EdgeCut can reduce content access latency by up to 30% compared to a recent solution that requires three times the storage resources. Additionally, EdgeCut reduces the cloud access rate by 20% to 50%. Notably, EdgeCut exhibits low overhead when deployed on edge servers.

ACKNOWLEDGMENTS

We sincerely thank our anonymous shepherd and reviewers for their insightful suggestions. Y. Liu, S. Shi, M. Wang, and C. Qian were partially supported by NSF Grants 2322919, 2114113, 1932447, and 1750704. Y. Wang was supported by NSF grant 2118745.

REFERENCES

[1] [n. d.]. https://cloudlab.us/.
[2] [n. d.]. https://seattle.poly.edu/.
[3] [n. d.]. https://www.planet.com/.
[4] [n. d.]. https://www.statista.com/statistics/744126/facebook-user-posts-per-month/.
[5] [n. d.]. Redis. https://github.com/redis/redis.
[6] [n. d.]. ZeroMQ. https://github.com/zeromq/libzmq.
[7] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. 2017. Mobile edge computing: A survey. *IEEE Internet of Things Journal* 5, 1 (2017), 450–465.
[8] J Almeida, AZ Broder, P Cao, and L Fan. 1998. A scalable wide-area web cache sharing protocol. *SIGCOMM98* (1998).
[9] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload Analysis of a Large-Scale Key-Value Store. In *In Proc. of ACM SIGMETRICS*.
[10] Andrei Broder, Michael Mitzenmacher, and Andrei Broder I Michael Mitzenmacher. 2002. Network applications of bloom filters: A survey. In *Internet Mathematics*. Citeseer.
[11] I-R Chen and Baoshan Gu. 2003. Quantitative analysis of a hybrid replication with forwarding strategy for efficient and uniform location management in mobile wireless networks. *IEEE Transactions on Mobile Computing* 2, 1 (2003), 3–15.
[12] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. 2014. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. 75–88.
[13] Ivan Farris, Tarik Taleb, Miloud Bagaa, and Hannu Flick. 2017. Optimizing service replication for mobile delay-sensitive applications in 5G edge network. In *2017 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.
[14] Ivan Farris, Tarik Taleb, Antonio Iera, and Hannu Flinck. 2017. Lightweight service replication for ultra-short latency applications in mobile edge networks. In *2017 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.
[15] Daniel Fernholz and Vijaya Ramachandran. 2007. The k-orientability thresholds for G n, p. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Citeseer, 459–468.
[16] Frédéric Gabry, Valerio Bioglio, and Ingmar Land. 2016. On energy-efficient edge caching in heterogeneous networks. *IEEE Journal on Selected Areas in Communications* 34, 12 (2016), 3288–3298.
[17] Maria Kihl, Robin Larsson, Niclas Unnervik, Jolina Haberkamm, Ake Arvidsson, and Andreas Aurelius. 2014. Analysis of Facebook content demand patterns. In *2014 International Conference on Smart Communications in Network Technologies (SaCoNeT)*. IEEE, 1–6.
[18] Darwin Klingman, Albert Napier, and Joel Stutz. 1974. NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. *management science* 20, 5 (1974), 814–821.
[19] Jun Li, Hao Wu, Bin Liu, Jianyuan Lu, Yi Wang, Xin Wang, Yanyong Zhang, and Lijun Dong. 2012. Popularity-driven coordinated caching in named data networking. In *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*. 15–26.
[20] Zaoxing Liu, Zhihao Bai, Zhenming Liu, Xiaozhou Li, Changhoon Kim, Vladimir Braverman, Xin Jin, and Ion Stoica. 2019. {DistCache}: Provable Load Balancing for {Large-Scale} Storage Systems with Distributed Caching. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*. 143–157.
[21] Bruce M Maggs and Ramesh K Sitaraman. 2015. Algorithmic nuggets in content delivery. *ACM SIGCOMM Computer Communication Review* 45, 3 (2015), 52–66.
[22] Najme Mansouri and Mohammad M Javidi. 2018. A new prefetching-aware data replication to decrease access latency in cloud environment. *Journal of Systems and Software* 144 (2018), 197–215.
[23] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. 2017. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials* 19, 4 (2017), 2322–2358.
[24] Joseph Noor, Mani Srivastava, and Ravi Netravali. 2021. Portkey: Adaptive key-value placement over dynamic edge networks. In *Proceedings of the ACM Symposium on Cloud Computing*. 197–213.
[25] Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo hashing. *Journal of Algorithms* 51, 2 (2004), 122–144.
[26] Jinglei Ren. 2016. YCSB-C. https://github.com/basicthinker/YCSB-C.
[27] B. Schlinker et al. 2017. Engineering Egress with Edge Fabric: Steering Oceans of Content to the World. In *Proc. of ACM SIGCOMM*.
[28] Shouqian Shi and Chen Qian. 2020. Ludo hashing: Compact, fast, and dynamic key-value lookups for practical network systems. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 2 (2020), 1–32.
[29] Narayanan Shivakumar, Jan Jannink, and Jennifer Widom. 1997. Per-user profile replication in mobile environments: Algorithms, analysis, and simulation results. *Mobile Networks and Applications* 2, 2 (1997), 129–140.
[30] Narayanan Shivakumar and Jennifer Widom. 1995. User profile replication for faster location lookup in mobile environments. In *Proceedings of the 1st annual international conference on Mobile computing and networking*. 161–169.
[31] Ao-Jan Su, David R. Choffnes, Aleksandar Kuzmanovic, and Fabian E. Bustamante. 2009. Drafting Behind Akamai: Inferring Network Conditions Based on CDN Redirections. *IEEE/ACM TRANSACTIONS ON NETWORKING* (2009).
[32] Tian Wang, Lei Qiu, Arun Kumar Sangaiah, Anfeng Liu, Md Zakirul Alam Bhuiyan, and Ying Ma. 2020. Edge-computing-based trustworthy data collection model in the internet of things. *IEEE Internet of Things Journal* 7, 5 (2020), 4218–4227.
[33] Shiow-yang Wu and Yu-Tse Chang. 2006. A user-centered approach to active replica management in mobile environments. *IEEE Transactions on Mobile Computing* 5, 11 (2006), 1606–1619.
[34] Xiaoyu Xia, Feifei Chen, Qiang He, John Grundy, Mohamed Abdelrazek, and Hai Jin. 2020. Online collaborative data caching in edge computing. *IEEE Transactions on Parallel and Distributed Systems* 32, 2 (2020), 281–294.
[35] Junjie Xie, Deke Guo, Xiaofeng Shi, Haofan Cai, Chen Qian, and Honghui Chen. 2020. A fast hybrid data sharing framework for hierarchical mobile edge computing. In *IEEE INFOCOM*. IEEE, 2609–2618.
[36] Junjie Xie, Chen Qian, Deke Guo, Xin Li, Shouqian Shi, and Honghui Chen. 2019. Efficient Data Placement and Retrieval Services in Edge Computing. In *IEEE ICDCS*. IEEE.
[37] Junjie Xie, Chen Qian, Deke Guo, Minmei Wang, Shouqian Shi, and Honghui Chen. 2019. Efficient indexing mechanism for unstructured data sharing systems in edge computing. In *IEEE INFOCOM*. IEEE, 820–828.
[38] Jianliang Xu, Bo Li, and Dik Lun Lee. 2002. Placement problems for transparent data replication proxy services. *IEEE Journal on Selected areas in Communications* 20, 7 (2002), 1383–1398.
[39] Juncheng Yang, Yao Yue, and K. V. Rashmi. 2021. A Large-scale Analysis of Hundreds of In-memory Key-value Cache Clusters at Twitter. *ACM Transactions on Storage* (2021).
[40] Kok-Kiong Yap et al. 2017. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering. In *Proc. of ACM SIGCOMM*.
[41] Ye Yu, Djamal Belazzougui, Chen Qian, and Qin Zhang. 2018. Memory-efficient and ultra-fast network lookup and forwarding using Othello hashing. *IEEE/ACM Transactions on Networking* 26, 3 (2018), 1151–1164.
[42] Yuming Zhang, Bohao Feng, Wei Quan, Aleteng Tian, Keshav Sood, Youfang Lin, and Hongke Zhang. 2020. Cooperative edge caching: A multi-agent deep learning based approach. *IEEE Access* 8 (2020), 133212–133224.
[43] Rui Zhu, Bang Liu, Di Niu, Zongpeng Li, and Hong Vicky Zhao. 2016. Network latency estimation for personal devices: A matrix completion approach. *IEEE/ACM Transactions on Networking* 25, 2 (2016), 724–737.