StarCDN: Moving Content Delivery Networks to Space

William X. Zheng
University of Illinois Urbana-Champaign

Aryan Taneja
University of Illinois Urbana-Champaign

Maleeha Masood
University of Illinois Urbana-Champaign

Anirudh Sabnis
Akamai Technologies

Ramesh K. Sitaraman
University of Massachusetts Amherst
Akamai Technologies

Deepak Vasisht
University of Illinois Urbana-Champaign

Abstract

Low Earth Orbit (LEO) satellite networks, such as Starlink, provide global internet access and currently serve content to millions of users. Recent work has shown that existing network infrastructures, such as Content Delivery Networks (CDNs), are not well-suited to satellite network architectures. Traditional terrestrial CDNs degrade performance for satellite network users and do not alleviate the congestion in the ground-satellite links. We design StarCDN, a new CDN architecture that caches content in space to improve user experience and reduce ground-satellite bandwidth usage. The fundamental challenge in designing StarCDN lies in the orbital motion of satellites, which causes each satellite's coverage area to change rapidly, serving vastly different regions (e.g., US and Europe) within minutes. To address this, we introduce new consistent hashing and relayed fetching schemes tailored to LEO satellite networks. Our design enables cached content to *flow* in the opposite direction of the orbital motion to counter satellite motion. We evaluate StarCDN against multiple baselines using real-world traces from Akamai. Our evaluation demonstrates that StarCDN can reduce the ground-to-satellite bandwidth utilization by 80% and improve user-perceived latency by 2.5X. Further, we make available an open-source trace generator, SpaceGEN, for realistic simulations of satellite-based CDNs.

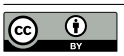
CCS Concepts

• **Networks** → **Network services**; **Mobile networks**; • **Computing methodologies** → **Distributed computing methodologies**.

Keywords

Starlink, LEO satellite networks, content delivery networks, network trace generation.

ACM Reference Format:
William X. Zheng, Aryan Taneja, Maleeha Masood, Anirudh Sabnis, Ramesh K. Sitaraman, and Deepak Vasisht. 2025. StarCDN: Moving Content Delivery Networks to Space. In *ACM SIGCOMM 2025 Conference (SIGCOMM '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3718958.3754345



SIGCOMM '25, Coimbra, Portugal

ACM ISBN 979-8-4007-1524-2/25/09
https://doi.org/10.1145/3718958.3754345

1 Introduction

Satellite networks, offered through mega constellations operating in Low Earth Orbits (LEO), are rapidly gaining traction. Starlink, a leading LEO satellite network provider, already has over four million subscribers across 100+ countries [53]. It currently operates more than 7,000 satellites, with plans to expand to 40,000 satellites [10] and to provide direct-to-cell services [52]. Similarly, other companies such as OneWeb and Amazon Kuiper offer/plan to offer similar LEO satellite-based networking services. Increasingly *LEO satellite networks (LSNs)* are used for delivering content to users around the globe [53]. *Our work is focused on enhancing the performance and reducing the cost of content delivery using LSNs.*

Content Delivery Networks (CDNs): CDNs were invented a quarter century ago to enhance the performance and reduce the cost of delivering content such as websites, media and downloads over the terrestrial internet [20]. CDNs deploy hundreds of thousands of "edge" servers around the world to cache and serve content from locations that are "proximal" to the user [42]. CDNs enable content to be served with lower latency, i.e., higher performance as perceived by the user, as content traverses a shorter network path from a proximal edge server to the user. CDNs also reduce the cost and network utilization since content can be downloaded once to a CDN edge server and delivered multiple times to users, saving the upstream WAN bandwidth (called midgress [59]) of transmitting content from an origin server to the edge. To achieve the performance and cost benefits, CDNs deploy clusters of edge servers across the globe, where each cluster caches and serves content to a proximal set of users. Further, CDNs use sophisticated techniques such as consistent hashing to manage the content within these clusters [36]. The benefits that CDNs provide have made them an essential component in modern internet infrastructure, and CDNs serve nearly 75% of global internet traffic [15].

Shortcomings of the current state-of-the-art: Current solutions that use traditional (terrestrial) CDN technology in conjunction with LSNs have several shortcomings that motivate our work. Recent work [8, 9] has shown that using traditional terrestrial CDNs in conjunction with Starlink degrades the Quality of Experience (QoE) for users. This degradation occurs because user traffic in Starlink flows through a bent-pipe architecture, wherein a user connects to a satellite, which in turn connects to the nearest ground station, as shown in Fig. 1. The ground station, then, connects to an edge server of the terrestrial CDN, increasing the latency by over 100 milliseconds [27]. A small fraction of the network traffic may also flow through inter-satellite links (ISLs) – ISLs have abundant bandwidth (100Gbps) compared to ground-satellite links (20Gbps).

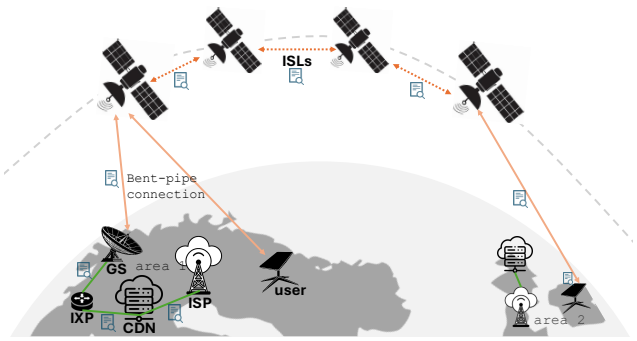


Figure 1: *Satellite network users often experience higher latencies compared to terrestrial users when accessing content due to the bent-pipe architecture where a user connects to a CDN's edge server via one more satellites and a ground station (GS), traversing ground-to-satellite links and perhaps also inter-satellite links (ISLs).*

In such cases, a user in area-1 (Fig. 1) may connect via ISL to a CDN edge server in area-2 that is much farther than their home location. Besides the increased latency, the CDN server in area-2 may also cache different content and apply different geo-access constraints. For example, [40] observed that users in Africa may connect to ground stations in Europe that connect to CDN servers that likely have Europe-specific content in their cache.

In addition to performance degradation due to the increased latency, traditional CDNs do not improve the utilization of the satellite links of the LSNs. In the current state-of-the-art, if multiple users watch the same video, it must be uploaded multiple times to the satellite, wasting precious uplink bandwidth to the satellite.

Our focus: *The primary question that motivates our work is whether CDN technology can be used to enhance performance, improve network utilization, and reduce the operating cost of delivering content to users of LSNs.* Our main approach to answering this question is exploring whether a system of edge servers can be deployed in LEO satellites to cache and serve content to (terrestrial) users. However, the significant challenge to pursuing this approach is the fact that the edge servers are in motion as the satellite orbits the Earth once every 90 minutes. Unlike a traditional CDN where the edge servers are stationary, in our setting, the users that are proximal to an edge server vary dynamically in the order of minutes, as do their content access patterns. *Rethinking the CDN's edge cluster architecture in the new context of LSNs, in particular its content placement, request routing, and caching, is the main contribution of our work.*

Our approach: We propose a space-based content delivery network called StarCDN that is specifically architected to work with LSNs. StarCDN leverages emerging computational capabilities of satellites to deploy edge servers that can cache content in space, thus reducing the latency of access for users and improving their quality of experience. Further, StarCDN reduces the uplink bandwidth required for satellite networks, allowing the scarce spectrum to be repurposed for downlink demand. In designing StarCDN, we solve three key challenges:

(i) Multi-satellite Redundancy: To provide optimal service at long range, LSNs use dense deployments comprising thousands of satellites. Therefore, at any time instant, a Starlink user can connect to 10+ satellites. This set of satellites is dynamic and can change within a few minutes due to the satellite's orbital motion. Since a user can connect to any of the visible satellites, the content requested by this user must be cached at *all* the visible satellites for the user to reliably receive it from the cache. This leads to high miss rates and wastage of precious storage space on satellites.

To counter these effects, we propose a *consistent hashing* scheme [30, 36] for edge servers deployed in the satellites. In StarCDN, we hash each object to one of K (e.g., $K = 4$) buckets and map each bucket to a different satellite. When a satellite receives a request for data in its own bucket, it can simply respond with the data (i.e., cache hit) or request data from the ground (i.e., cache miss). However, if a satellite receives a request for data from a different bucket, it forwards the request to a neighboring satellite with this bucket using its ISL link. We map buckets to satellites in a grid pattern such that each bucket is *at most* $2\lfloor \frac{\sqrt{K}}{2} \rfloor$ hops away in the constellation. Our consistent hashing approach optimizes the utilization of storage capacity in space, while minimizing the additional latency experienced for requests.

(ii) Orbital motion: In terrestrial CDNs, objects are cached in servers that are located in physical proximity to the users. Caching decisions rely on local popularity characteristics of objects served by the CDN. For example, least recently used (LRU) or least frequently used (LFU) objects are removed from the cache to make way for the addition of new objects. However, LEO satellites orbit around the Earth at speeds of around 8 km per second and serve a given location for less than ten minutes. A satellite serving users in the United States may serve European users in a matter of minutes. Therefore, the access pattern, popularity statistics, and cache content rapidly grow stale, leading to low hit rates. Our analysis shows that a simple LRU cache deployed on satellites will achieve a hit rate below 60% due to such orbital motion.

To counter the effects of orbital motion, StarCDN deploys a *relayed fetch* technique, where a satellite can relay a request to a neighboring satellite (with the same bucket ID) using its ISL link in case of a cache miss. This allows StarCDN to benefit from a previous satellite's cached object that just served a region. Effectively, this allows cache content to flow backwards, i.e., in the opposite direction of the orbital motion. Note that this backwards flow does not propagate objects that are no longer popular because the relay is initiated only in cases of a cache miss of an accessed object. We limit the relay to the nearest neighbor to cap the additional latency incurred due to such relays.

(iii) A novel publicly available trace generator that captures the content access patterns of LSN users distributed around the world: It is challenging to evaluate a space-based CDN design due to the requirement for globally distributed traffic traces that capture content accesses of users around the world. Traditional CDN designs are evaluated using traces from one (or a few) locations. However, a satellite orbits the globe, and to effectively evaluate StarCDN, we require traces from multiple locations on Earth for a large period of time. To achieve this, we first collected limited real-world traffic traces from nine locations across the world from Akamai's CDN

over one day. Then, we designed a new synthetic trace generator, SpaceGEN, that uses footprint descriptors [58] to capture both temporal variations of content access patterns within a location, such as object access frequencies, and geographical variations of content accesses across locations, such as how content accessed across locations overlap. Further, SpaceGEN can produce synthetic traces for the major traffic classes hosted on a CDN, such as web, video and software downloads. Our trace generator enables realistic long-term evaluation of StarCDN and other baselines.

Summary of contributions:

- StarCDN is the first space-based CDN designed to improve the content access experience of LSN users while optimizing the utilization of the ground-satellite network (§3).
- We design and evaluate a novel LSN-specific consistent hashing scheme to reduce redundancy in satellite caches and improve cache performance (§3.2). Further, we design and evaluate a relayed fetch scheme for content to counter the effects of the satellite's orbital motion (§3.3).
- StarCDN is evaluated using realistic content access traces from a geo-distributed set of LSN users. These synthetic traces were derived using SpaceGEN, our novel trace generator that produces synthetic traces that are similar to actual production traces for different traffic classes, such as videos, web, and download content (§4). *To the best of our knowledge, SpaceGEN is the first trace generator for cache simulations that incorporates both temporal and geographic variations in content access patterns of users.* To support more research in the area, we have released both our StarCDN simulation framework[1] and SpaceGEN trace generator[2] on GitHub.
- We simulated CDN edge servers on satellites using actual servers while simulating the satellite orbital motion and field of views using the Microsoft CosmicBeats [38, 48] simulator (§5). We simulate 1170 satellites from the Starlink constellation for our experiments. Our experiments demonstrate that StarCDN can improve the cache hit rates in space from 60% to 75%. It can also reduce the satellite network uplink utilization by up to 80%, and improve user-perceived latency by 2.5X.

2 Background

We provide a brief background for LSNs and CDNs.

2.1 LEO Satellite Networks

Satellite-based internet, using LEO satellites, has gained widespread adoption over the last decade, with the emergence of Starlink, Oneweb, and Kuiper[2, 43, 54]. Starlink is the most mature LEO Network service provider, with more than 7,000 satellites in orbit.

Orbital motion: LEO satellites orbit around the Earth at an altitude of around 550 km (compared to traditional geostationary satellites at nearly 36000 km). Therefore, LEO satellites offer significantly lower propagation delay to support modern internet applications. LEO satellites also offer a better link bandwidth budget (and hence more throughput) due to their proximity to Earth. However, LEO satellites must have high speeds in order to maintain their orbits. For context, LEO satellites orbit the Earth approximately every 90 minutes. Due to this fast orbital motion, a user dish can connect to a satellite for at most a few minutes, resulting in frequent user-satellite link "handovers" when satellites move out of view.

	Avg Delay(ms)	Std Delay(ms)	Min Delay(ms)	Bandwidth (Gbps)
Intra-orbit ISL	8.03	0.376	4.76	100
Inter-orbit ISL	2.15	0.492	1.32	100
GSL	2.94	1.01	1.82	20

Table 1: *Propagation delay and bandwidth of Starlink links.*

Network routing: Starlink satellites are equipped with radio wave antennas for ground-satellite links and optical transceivers for inter-satellite links (ISLs) [55]. The common mode of operation is a *bent pipe* model, where the user dish connects to a satellite, which in turn connects to a Starlink ground station. The ground station forwards packets through the terrestrial network and connects to the rest of the internet through an Internet Exchange Point (IXP). This pipeline is shown in Fig. 1.

ISLs have recently been introduced in Starlink and are commonly used in locations without a ground station nearby (e.g., in many countries in Africa). A satellite uses ISLs to route traffic to other satellites, which downlink it to a ground station on Earth. Starlink satellites typically support four ISLs: two *intra-orbit* links (to previous and next satellites in the same orbit) and two *inter-orbit* links (to adjacent satellites in parallel orbits)[72]. We list propagation delays and bandwidths of ground-satellite links (GSLs) and ISLs in Table. 1. We visualize Starlink's orbits and ISLs in Fig. 5b.

2.2 Content Delivery Networks

A Content Delivery Network (CDN) is a large distributed system potentially consisting of hundreds of thousands of edge servers deployed in thousands of locations across the world [20, 42]. The edge servers are deployed in clusters where each cluster is deployed within a data center or colocation facility. A CDN edge server can cache and deliver content to users on behalf of potentially thousands of content providers. While the original content provider stores all objects in their "origin" servers, CDNs do not cache every object at the edge server due to limited cache space. When a user requests content, say a web page or a video, that request is routed to a proximal edge server of the CDN. If that server has the requested content, a *cache hit* is said to have occurred, and the user is served the content. If the edge server does not have the requested content, a *cache miss* is said to have occurred, and the content is fetched over the WAN from the origin server and served to the user. When the cache of an edge server is full, CDN providers utilize eviction policies to determine which content should be removed from the cache[14, 71]. Various eviction policies have different strengths and weaknesses. A commonly used policy is the Least Recently Used (LRU) policy, and different LRU variants are often deployed in commercial CDNs [36] for their simplicity and effectiveness. In LRU, an object with the oldest last access is evicted from the cache.

Traditional CDN infrastructures are geographically static, relying on deploying edge servers close to the users [25] and routing requests from users to a proximal edge server [13, 36] that can

[1] https://github.com/ConnectedSystemsLab/StarCDN-Simulator
[2] https://github.com/ConnectedSystemsLab/SpaceGEN

serve the content. The cache hit rate is a key metric to maximize because each cache miss increases both the latency experienced by the user and the upstream WAN bandwidth for fetching the "missed" content from the origin server. The hit rate can be measured in two ways – the *request hit rate* is the fraction of requests that were cache hits, while the *byte hit rate* is the fraction of bytes served for requests that were cache hits.

CDN architectures vary from provider to provider. Netflix [26, 27], YouTube [1, 67], and Amazon Prime are vertically integrated and provide both the content and CDN services for their users and are largely focused on video content delivery. On the other hand, Akamai [42, 47] and Cloudflare [21, 34] provide general-purpose third-party CDN services for a large number of content providers, delivering a range of traffic classes such as web, videos, and software downloads. While some CDNs serve content from larger edge server clusters in fewer locations, others deploy smaller clusters in a large number of locations. However, all CDNs share the basic principles of routing requests from users to proximal edge servers that cache and serve popular content. While we use traces from Akamai's CDN for our empirical evaluation, the architectural ideas for caching and content management proposed and studied in this work are applicable across a wide range of CDN architectures. Further, our open-sourced trace generation tool (SpaceGEN) and simulation framework allow evaluation of other satellite-based CDN architectures using production logs from other CDNs.

2.3 Feasibility of In-space Compute and Storage

CDN edge servers in space will require storage, compute, and power on satellites. We discuss the feasibility of providing these resources on board a satellite. Recent developments in the last decade have shown significant potential for placing computational and storage capabilities on satellites in space [7, 19, 41, 63, 70]. In-orbit computing [7] explores the feasibility of augmenting satellites with edge-compute capabilities, a constraint relevant for CDN integration. The same work describes the power, weight, and voluminous feasibility of placing a high-end server with up to 2 TB storage on Starlink satellites. Newer servers can hold even more storage and compute with the same requirements, indicating that lesser volume and weight will be required with advancing technologies in the field. Some missions have already launched satellites with computing capabilities in space for various applications [22, 32, 60, 69]. For example, both Planet [32] and European Space Agency [22] have demonstrated the ability to run machine learning models onboard small satellites using edge-computing devices (such as the NVIDIA Jetson).

3 StarCDN System Architecture

We propose a novel space-based content delivery network, StarCDN, that is specifically designed to work with LSNs. StarCDN deploys CDN edge servers in the satellites to cache content while utilizing the ISL links to fetch content from neighbors as needed. In designing StarCDN, we aim for the following objectives:

- **Reduce latency:** Satellite network users should experience similar latencies as terrestrial network users for accessing content.

	Britain	Germany	Turkey
Britain	100%	11%(49%)	2%(15%)
Germany	16%(45%)	100%	4%(31%)
Turkey	23%(37%)	34%(72%)	100%

Table 2: *Percent of objects (traffic) accessed by users in the first country (row) that are also accessed in the second country (column). The content overlap is low since users in these European countries speak different languages and seldom access the same content.*

- **Reduce uplink bandwidth utilization:** Currently, all content served to users must be sent from ground stations to satellites via their uplinks. We aim to reduce uplink utilization by reducing the need to fetch content from the ground. This goal is motivated by the surge in demand for satellite-based services. Over the past few years, the Starlink user base has grown from approximately 1 million to over 4 million globally [50, 56], leading to increased contention for uplink and downlink bandwidth, particularly in densely populated and high-traffic regions. In response, Starlink has started to pause new subscriptions in areas of high demand [16]. Reducing uplink utilization will free up bandwidth that can be repurposed for additional users.
- **Compatibility with current network architecture:** We aim to maintain compatibility with existing satellite network architecture, i.e., we carefully model GSLs, ISLs, constellation size, satellite orbits, and ground stations using publicly available information about the Starlink network.

3.1 A Naive Design and Resulting Challenges

To motivate the design choices in StarCDN, we begin by considering a naive satellite CDN design. Consider a constellation of Starlink satellites where each satellite is equipped an edge server that can cache content. Each cache follows the popular LRU eviction policy. This setup mimics a terrestrial CDN design where each server independently implements an LRU-like eviction policy. We analyze the unique challenges that satellite networks pose for this setup.

3.1.1 Challenge 1: Dynamic Access Patterns due to Orbital Motion. A LEO satellite orbits around the Earth in 90 minutes. Since the Earth rotates around its axis every 24 hours (not synchronized with the LEO satellite), the LEO satellite covers different parts of Earth during each orbit (see Fig. 3 for an illustration). This implies that each cache serves users in different geographies over time. We note that this is a drastically different scenario compared to terrestrial CDNs, where servers are stationary and typically serve users in an area proximal to the server.

To understand the implications of this motion, we analyze the geographic diversity in content access patterns. Specifically, we collect and analyze production traffic traces of users accessing video content from nine edge server clusters across the globe of the Akamai CDN: Mexico City, Dallas, Atlanta, Washington D.C., New York City, London, Frankfurt, Vienna, and Istanbul. We sample traces (subsampled at 1%) from diverse cities around the world that have a large population and user demand. The traces contain anonymized information about each access made by users, including what content was accessed at what time and from which edge server. We

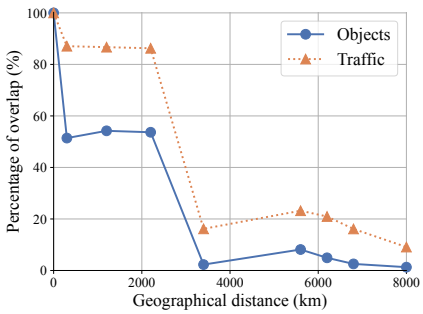


Figure 2: *Percent of objects and traffic accessed by users in New York that are also accessed by users at a different location. The overlap decreases with the geographic distance of the location from New York.*

sample more cities from the US because it has the highest Starlink users today[18]. The whole trace consists of 423M video requests (512TB) for 24M unique objects (24TB).

To quantify the impact of satellite motion, we study the overlap between objects and the content access traffic from each pair of countries with different official languages in Europe in Table. 2. We note that different languages create noticeable traffic diversity even within a single continent. These diverse regions can be traversed and served by the same satellite within minutes.

Further, we plot the overlap in terms of unique objects and unique volume of access traffic with respect to geographical distance to New York in Fig. 2. Our analysis reveals several interesting insights. First, for regions closer to New York (< 3000km), the overlap is around 55% in terms of the objects served. This indicates that even close cities like New York and Washington DC have non-overlapping objects being requested by users. However, in terms of traffic volume, around 90% of the traffic volume goes to objects that are available in both adjacent locations. Second, across countries that are more than 3000km apart, the overlap is fairly low both in terms of unique objects served and the traffic volume. Even in an English-speaking city like London, only about a quarter of the traffic is also present in New York. This indicates high geographic diversity. Traditionally, a CDN provider would provision dedicated cache clusters for geographically distinct regions. However, the same satellite can go from US to Europe within tens of minutes, indicating the unique challenge of designing caching schemes for satellites. By the time a satellite sees enough traffic to build a cache, its orbital motion pulls it away to a different region.

Takeway: *Rapid orbital motion of LEO satellites places them over different geographic regions within minutes. These regions require different content cached on the satellite.*

3.1.2 Dynamic client-server relationships. A corollary of the orbital motion discussed above is that client-server relationships constantly evolve over time in satellite networks. In terrestrial CDNs, a mapping system is used to route a client's (i.e., user's) request to a proximal edge cluster, either using the DNS system [13, 36] or IP anycast [11, 73], and an intra-cluster "local" load balancer assigns the client to an edge server within the chosen cluster. The server assignment of a client is generally stable since the server is stationary and the clients are generally located within the same geographical

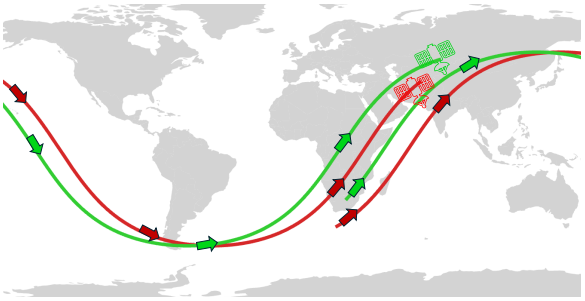


Figure 3: *Trajectory of two satellites, three parallel orbits away.*

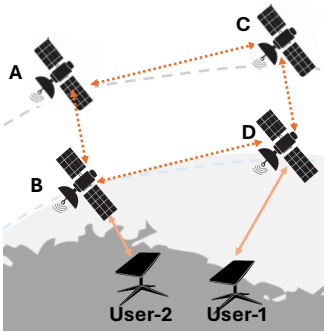


Figure 4: *Two users with four satellites in view. Each user is connected to one of these satellites by Starlink's satellite link scheduler.*

area. While a client may get assigned to multiple servers over time due to changes in server and network conditions, these servers typically cache the same regional content relevant to the clients.

However, LEO satellites orbit around the Earth at low altitudes, and each of them can only cover a small region of Earth as compared to GEO satellites. Satellite network providers usually build a large satellite constellation to increase the coverage area. For example, at any time instant, a Starlink client often has 10+ satellites in view, and the Starlink scheduler is responsible for scheduling client-to-satellite links [51]. We refer to the satellite scheduled for a client as the *first contact satellite*, which changes rapidly. Recent literature shows that the Starlink scheduler reconfigures the user-satellite mapping every 15 seconds [51]. In general, in any LEO network, the client-satellite mapping cannot last beyond a few minutes. This means in satellite-based CDNs, it is impossible to maintain a stable client-server mapping beyond few minutes. There are two implications of this dynamic mapping. Multiple satellites that serve the same area may need to cache and serve the same objects redundantly, resulting in reduced cache storage efficiency. Further, satellites need to evolve their cached content as the geographical area changes.

Consider an example in Fig. 4. user-1 and user-2 are scheduled to choose satellites D and B respectively, as the first contact satellites, decided by the satellite link scheduler (all satellites in this figure are in view and connectable for both users). Now, user-1 initiates a request, which is resolved to the proximal edge server on satellite D. Satellite D checks its local cache and serves the content if it's available. Otherwise, it downlinks the request to terrestrial networks to retrieve and cache the requested content in its local cache. What if user-2 now requests the same content as user-1? The same process

will happen, and the cache server on satellite B will fetch and cache the same content, without recognizing that this request can be fulfilled by retrieving the content from the cache at the neighboring satellite D through ISL links. Thus, the redundant storage of the same content in both satellites decreases the overall cache storage efficiency of the system, and user-2 may perceive high latency if there is a cache miss at satellite B, resulting in ground-satellite communications.

Takeaway: *Dynamic client-server mappings may lead to redundant storage of the same content in multiple caches, resulting in a reduction of the overall cache storage efficiency. Further, it could also result in greater latency during cache misses due to additional ground-satellite communication.*

3.2 StarCDN: Consistent Hashing to Reduce Redundant Caching

To counter the dynamic-client server relationships, we propose a consistent hashing scheme for partitioning cache content across satellites. Consistent hashing is widely used in production storage systems to provide load-balancing and fault tolerance [30, 36, 71]. Conceptually, consistent hashing involves hashing both servers and objects to a unit circle. Each object is mapped to the next server that appears clockwise on that unit circle. This allows the objects to be divided among servers on a single cluster, reducing redundancy. In terrestrial networks, consistent hashing is done across servers located close to each other within a cluster as described in [36].

We cannot apply consistent hashing to the satellite use case as is. Specifically, LEO satellites are typically small and do not support many servers running simultaneously under the limited power, size, and thermal constraints [7]. Moreover, even if we were to map objects to different servers on a satellite, it would not solve the redundant caching problem discussed before.

StarCDN proposes a satellite-specific variant of consistent hashing. Similar to standard consistent caching, we partition the objects into K disjoint buckets. Subsequently, each bucket is then mapped to a *different* satellite. But how do we identify which bucket should be mapped to which satellite? Formally, this problem can be mapped to a graph coloring problem [29] for an arbitrary constellation topology, with constraints imposed by the presence of ISLs and latency requirements to fetch content from different satellites.

The Starlink topology is shaped in a grid pattern, which simplifies the mapping problem. Inter-satellite links naturally lead to a grid topology of satellites. Each satellite typically connects to four other satellites (front, back, left, right). We map the K buckets on this grid in a repeating $\sqrt{K} \times \sqrt{K}$ pattern. For example, we can map 9 buckets in a 3×3 pattern.

An example for $K = 4$ is shown in Fig. 5a where the four buckets of content are stored in each 2×2 grid, for instance, the grid consisting of satellites S1, N1, S2, and N2 stores the $K = 4$ distinct buckets. Note that we cannot control which user talks to which satellite. This falls within the purview of Starlink's satellite link scheduler and has various constraints. However, when a user requests an object from its first-contact satellite (e.g., S1 in Fig. 5a), it can serve the object if the object is in its designated bucket. If not, it can compute the shortest path to a satellite with the object's bucket (say N1) and forward the request along that path. If this satellite (N1) has the object (cache hit), it forwards the object to the first-contact satellite (S1). If the satellite does not have the required object (cache miss), it can request it from the ground and store it in its cache for the future and also forward the object to the first-contact satellite (S1). The first-contact satellite (S1), then, forwards the object to the user.

Our scheme has several advantages. First, it increases the cache storage efficiency since each cache needs only cache $(1/K)^{th}$ of the objects, allowing more objects to be stored in space and increasing the cache hit rate. Second, it reduces redundancy as each object is only stored by the server that is assigned its bucket. If K satellites serve a single region, they do not need to cache the same content. Third, although there is an added latency to query an object from a neighboring satellite, all buckets are accessible in at most $2\left\lfloor \frac{\sqrt{K}}{2} \right\rfloor$ hops from the first-contact satellite. Finally, our consistent hashing scheme accommodates any cache replacement scheme within each server, including LRU (Least Recently Used), LFU (Least Frequently Used), Sieve [74], and others.

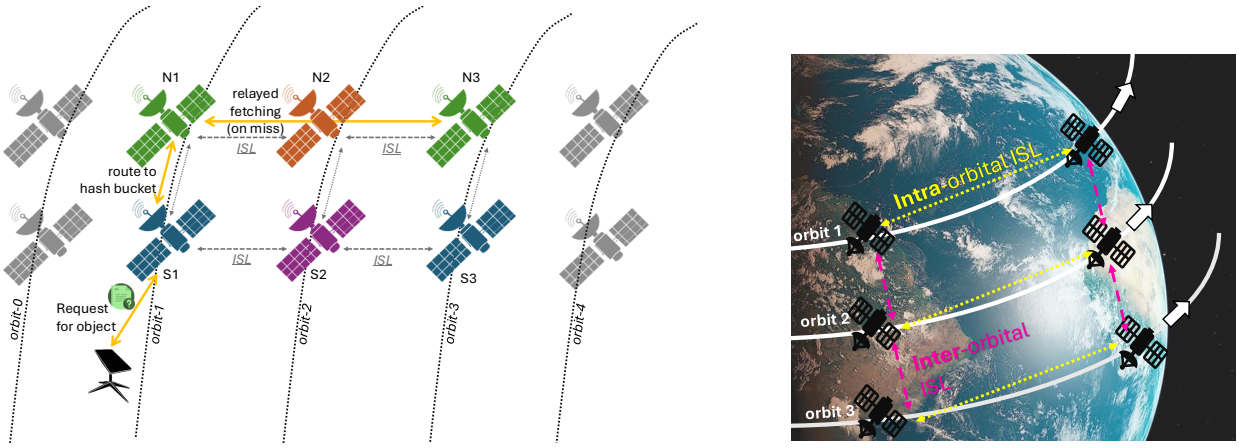
How do we choose K? The choice of K is driven by three factors. First, a small value of K increases the cache miss rate since a greater fraction of content is assigned to each cache and the larger content redundancy that is entailed. Second, a large value of K increases the latency to access an object. Third, there are constraints due to the constellation size, orbit design, and orbital motion. We find that values of $K = 4$ and 9 are generally compatible with the Starlink constellation. We evaluate this tradeoff in §5.3.

3.3 StarCDN: Relayed Fetch to Counter Orbital Motion

Next, we discuss how StarCDN addresses the challenge of dynamic shifts in the access pattern. Recall that the fundamental reason for such a dynamic access pattern shift is the orbital motion of a satellite. Unlike traditional CDNs, where the edge server remains in a single location and serves users proximal to that location, a LEO satellite is in rapid motion with respect to the surface of the Earth. Intuitively, our goal is to create a flow of cached content in the opposite direction of the orbital motion. This allows the cached content accessed by users at a particular location to stay above that location, while the server that caches the content keeps changing due to the orbital motion. We call this technique *relayed fetch*, where we allow a satellite to request objects from the neighboring satellites with the same bucket mapped to them. For example, in Fig. 5a, N1 can request content from N3 on its right since it is its next nearest neighbor with the same color/bucket. Such relays are only initiated in response to a cache miss at N1.

One concern of using relayed fetch is the latency overhead. Each one-way inter-orbital hop requires at least 2 ms, while an intra-orbital hop needs 8 ms. Given the added latency of intra-orbital links and the larger distance between them, StarCDN fetches data only from inter-orbital neighbors, avoiding intra-orbital neighbors to mitigate the high latency penalty associated with a cache miss.

To visualize inter-orbital links and the benefit of fetching from them, we visualize two satellite trajectories in Fig. 3. It shows the trajectory of two satellites three inter-orbit links away over one period. Note that the red satellite follows a path very similar to that of the green satellite (west inter-orbital neighbor of the red



(a) (b)

Figure 5: *(a) Consistent hashing routing and relayed fetching in StarCDN. (b) Orbital motion and ISLs of Starlink satellites.*

satellite) traveled in the previous period. This means a satellite's west neighbor has the historical footprint of requests we want to exploit.

Finally, we make a slight addition to our design to also allow satellites to fetch from their east inter-orbital neighbor (e.g., between N1 and N3), since this has the same latency penalty as fetching from the west neighbor. Our evaluation in §5 shows that this connection is less likely to be useful compared to the rightward links, but it doesn't incur any additional latency in StarCDN and hence, we choose to keep these links bidirectional.

Why not proactive prefetching? In our current design, we use relayed fetch in response to a *cache miss*. An alternative strategy is to proactively prefetch popular content from preceding satellites when entering a populous region. It can create a similar backflow of content to counter the effect of orbital motion. However, there is a risk of prefetching content that is *stale* and is no longer being requested by clients. We found this strategy to be less efficient than relayed fetch in terms of hit rate. While relayed fetch incurs an added latency, this latency only happens during the first request for a new object. Once the object has been fetched, it can be stored in the local cache. In contrast, if the proactively prefetched content is not used, it will be a waste of cache space at the receiver, a waste of power to transmit the data, and a waste of ISL bandwidth.

3.4 StarCDN: Robustness to Unavailability

StarCDN also consistently monitors the nearby satellite links and the reachability of bucket IDs. ISLs today can last weeks [37] with high stability. However, link unavailability is inevitable when satellites commence maneuvers to avoid collisions [35]. Note that such collision avoidance maneuvers are known in advance, as the satellite network operator needs to plan for them. A different type of failure is cache server unavailability, e.g., due to bringing down the server for a software update. Cache server unavailability is also common but transient [71].

StarCDN's response to such failures depends on whether the failure is transient or long-term (tens of minutes or hours). For transient failures, StarCDN simply reports a cache miss and forwards the requests to the ground. For the long-term failures, StarCDN's consistent hashing scheme remaps the bucket assigned to the unavailable satellite to the next available satellite (this satellite is now responsible for multiple buckets). Our approach temporarily leads to uneven loads, but the time duration of such failures is small enough that a larger reconfiguration of the consistent hashing scheme isn't needed. We evaluate our robustness to failures in §5.4.

4 SpaceGEN: Synthetic Trace Generator for Satellite-based CDNs

In this section, we describe our trace generation tool, SpaceGEN, that generates realistic synthetic traces of how LSN users access content. Our goal is to generate long-term synthetic traces suitable for simulating StarCDN and other satellite-based CDN designs from limited real-world traces collected from the production CDN. Our tool builds on the theory of footprint descriptors (FDs). FDs are traffic models that capture the manner in which users access content and were first proposed in [58]. FDs can predict cache hit rates for various traffic classes, such as videos, web, and downloads, and are used for cache provisioning in Akamai's production CDN. Further, variants of FDs were used to generate synthetic logs that are provably similar to production logs and incorporated in the synthetic trace generation tools TRAGEN [44] and JEDI [45]. *Unlike prior tools that generate synthetic traces for an individual cache at a specific location, SpaceGEN generates synthetic traces collectively across multiple locations that are suitable for simulating satellite-based CDNs.* We further show that the synthetic trace and the production trace yield similar hit rates when performing a cache simulation of a satellite traversing these locations. We have made this tool and the associated traffic models derived from the production CDN available to the research community to facilitate further research in the satellite-based CDN domain.

4.1 Traffic Models for Satellite-based CDNs

Our models capture essential statistics such as popularity, size, overlap, and access patterns of the objects requested by LSN users using the following traffic models: (i) Global Popularity Distribution (GPD), and (ii) Popularity-Size Footprint Descriptor (pFD). The GPD captures the correlation between objects and requests across

different locations, while the pFD describes the access patterns of objects from a single location. We compute the GPD and pFDs models from the production traces of the Akamai CDN. We have made these models available for public download.

Global Popularity Distribution (GPD). The GPD captures the joint distribution of an object's popularity and size across multiple locations. Formally, it is expressed as $P(p_1, \ldots, p_n, z)$, where p_i represents the popularity of the object at the i^{th} location, and z denotes the size of the object in bytes. Popularity p_i is defined as the number of requests made for the object in the production trace at location i. Unlike a normalized value, this definition of popularity is adopted by synthetic trace generation tools such as TRAGEN and JEDI. Thus, $P(p_1, \ldots, p_n, z)$ represents the probability that an object has size z and popularity p_i at location i.

Popularity-Size Footprint Descriptor (pFD). The pFD captures the request access patterns, popularity, and size of objects requested from a single location. It has been shown to capture: (i) Object-level properties: including the popularity distribution, size distribution, and request-size distribution, and (ii) Cache-level properties: such as request hit rate curves and byte hit rate curves of the trace. Formally, pFD is described as a probability distribution $P(p, z, s, t)$, where: (i) p and z represent the popularity and size of an object in the trace (ii) s denotes the number of unique bytes requested between consecutive accesses of an object (iii) t represents the inter-arrival time, i.e., the time between consecutive accesses of an object. The number of unique bytes, s, that are requested between consecutive accesses is known as the stack distance [45].

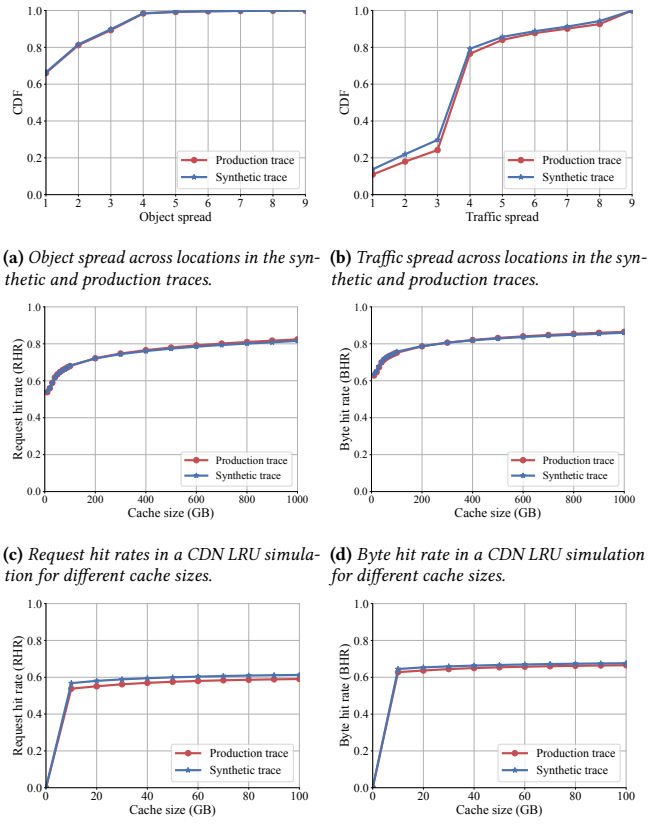
4.2 Trace Generation Algorithm

In this section, we present the synthetic trace generation algorithm used by SpaceGEN. The algorithm takes as input the GPD and the n pFDs derived from the production trace of each location. It generates n synthetic traces of user-specified lengths, with each trace corresponding to a specific location.

Initialization phase. We initialize an empty cache C_i corresponding to each location $i \in n$. To fill the caches, we iteratively create objects and assign them popularities and a size by sampling from the GPD. For each object o, the sample gives the popularity vector $\mathbf{p}$ and a size z where p_i denotes the popularity of the object in the i^{th} location. If $p_i > 0$, we add o to C_i. We repeat till each cache C_i, is at least as large as the maximum stack distance in the pFD of the i^{th} location.

Generation phase. We generate the n synthetic traces, each corresponding to a location $i \in n$, by the following procedure. First, for each location i, we compute $P_i(s|p, z)$ for all possible values of p and z from the i^{th} pFD. During each iteration of the trace generation phase, we examine the object at the top of the cache C_i. We add a request to the object in the synthetic trace. Let the object's popularity be p and its size be z. We then sample a stack distance s from $P_i(s|p, z)$. If the object has already received p requests in the synthetic trace, it is removed from the cache. Otherwise, it is removed from the top of the cache and reinserted within C_i at a stack distance s from the top. Finally, we assign timestamps to the traces based on either the average data rate derived from the pFD or a more fine-grained data rate computed from the real traces. We describe the algorithm in detail in Algorithm 1 in Appendix A.1.

4.3 Properties of the Synthetic Trace



(a) *Object spread across locations in the synthetic and production traces.*

(b) *Traffic spread across locations in the synthetic and production traces.*

(c) *Request hit rates in a CDN LRU simulation for different cache sizes.*

(d) *Byte hit rate in a CDN LRU simulation for different cache sizes.*

(e) *Request hit rate of a satellite equipped with an LRU cache for different cache sizes.*

(f) *Byte hit rate of a satellite equipped with an LRU cache for different cache sizes.*

Figure 6: *Comparison of synthetic and production traces.*

We will now show that the synthetic trace generated by Algorithm 1 is similar to the production trace. The *object spread* distribution and the *traffic spread* distribution of the synthetic and production traces are shown in Fig. 6a and Fig. 6b, respectively. Here, the object spread is the number of locations an object is accessed from and we observe that both the traces have similar object spreads. The traffic spread is the object spread weighted by the size and the number of requests made to the object. We observe that the traffic spread of the production and synthetic traces are similar.

In Fig. 6c (resp., Fig. 6d), we show that the request hit rates (resp., byte hit rate) of a cache simulation of a traditional CDN server using LRU yield similar results for the synthetic and the production trace. In particular, we observe an average difference of 0.4% in request hit rate (resp., 0.3% in byte hit rate) across all the cache sizes we simulated. Next, we simulate satellites in motion that are equipped with an LRU cache. In this case, we observe an average difference of 2% in request hit rate (resp., 1% in byte hit rate) between the synthetic and production traces across all cache sizes we simulate. We observe similar results when we simulate the StarCDN-Fetch architecture. Results are in Fig. 13c and Fig. 13d in Appendix A.1. Thus, we conclude that the synthetic traces can be used in lieu of the production traces for our evaluation.

5 Empirical Evaluation

We discuss our evaluation of StarCDN below.

5.1 Experimental Setup

CDN traces: To evaluate StarCDN over a sufficiently long period, we create 5-days-long synthetic traces using the SpaceGEN trace generator discussed in §4. SpaceGEN generates synthetic traces from traffic models derived from real-world Akamai production traces for the video traffic class described in §3.1. In total the synthetic traces for the video traffic class have 2 billion requests and 2.5PB content traffic. In addition, in §5.5, we also extend the evaluation to the web and download traffic classes by using SpaceGEN with the relevant traffic models for these classes.

Simulation setup: We collected up-to-date TLE data from CelesTrak [12] for Starlink-53-Gen-1 satellites' orbital information and orbital shell information from Starlink.sx[39]. Even though Starlink Gen1 satellites do not support ISLs, the Starlink Gen2 satellite constellation is still in the launch phase and not fully operational. Thus, we cannot obtain full orbital information from the Starlink Gen2 constellation. Instead, we use Starlink-53-Gen-1 as a representative of Starlink's constellation topology. We infer the ISLs to both inter-orbital and intra-orbital neighbors using the shell information. If the neighbor is out of service or out of slot, we assume the link cannot be established. We simulate 1,170 satellites in 72 orbits inclined at 53 degrees.

We implemented a trace-driven simulator composed of Microsoft's CosmicBeats [38, 48] simulator for the orbital motion, client link scheduling, and a multi-process cache replayer to mimic real-world asynchronous CDN accesses. For each CDN user node in CosmicBeats, we associate a synthetic trace from SpaceGEN that represents user requests for content from that geographic location. CosmicBeats determines the satellites available in view at each location and splits all requests within the discrete time step to different satellites. The time step of CosmicBeats is set to 15 seconds, aligned with the Starlink global scheduler's reconfigure interval [51]. CosmicBeats outputs logs of object access for every satellite, which are loaded into our cache replayer. The cache replayer spawns a process for each satellite that uses TCP to mimic ISLs. Finally, the replayer orchestrates the cache replay and allows satellite processes to simulate the real-world request traffic. We open-sourced the simulation framework, including the configuration files for running CosmicBeats. User can also generate their own configuration files for their dataset and new application logic.

Baselines: We compare StarCDN with two baselines. (a) **Naive LRU** places LRU caches on LEO satellites (as proposed in past work [7, 8]). We evaluate this baseline using the same simulation framework. (b) **Static Cache** is an idealized baseline, i.e., it assumes that there is no orbital motion and satellites stay static. The user-satellite mapping is static. This baseline is, in practice, unachievable. Yet, we plot this baseline as the north star for satellite-based CDNs.

5.2 Cache Hit Rate

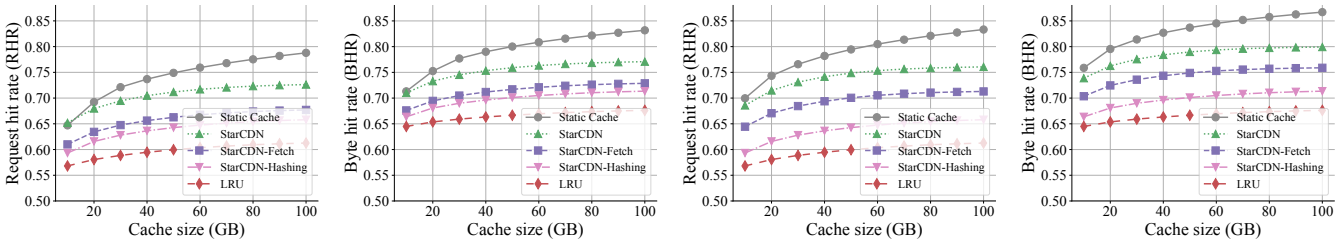
We run a simulation across multiple baselines for different cache sizes and plot the hit rate curves in Fig. 7. Past work [7] has reported the feasibility of placing a 2TB cache on satellites. Our traffic traces are sampled at 1% from production CDNs by objects. The production traces were collected for a duration of 1 day, and were collected at the same time for all the locations. We have accounted for time differences in each location by collecting traces from the locations that span the same 24 hours. These traces were collected from edge servers that serve video traffic. Since our traces' traffic rate is sampled by 1%, we also shrink the cache size in our experiment to reflect the hit rate when a larger traffic load is served by satellite-based CDN in the future. Therefore, our target cache size would be 20 GB. We vary the cache size from 10 to 100 GB to cover a span of configurations. In Fig. 7, we have five curves – one each for Static Cache, StarCDN, StarCDN-Fetch (StarCDN without relayed fetching), StarCDN-Hashing (StarCDN without hashing), and Vanilla LRU. We further study the performance of two different numbers of hash buckets $K = 4$ and $K = 9$.

First, we plot the request hit rate for $K = 4$ and $K = 9$ in Fig. 7a and Fig. 7c respectively. Request hit rate captures the percentage of requests being served from the cache. A higher hit rate implies better quality of service for the user (e.g., lower latency). As shown in the plot, StarCDN significantly outperforms the LRU baseline. At a cache size of 50 GB and $K = 4$, for example, LRU achieves a request hit rate of 60% and StarCDN achieves a request hit rate of 71%. This is because LRU finds it hard to deal with the dynamic data access patterns caused by the orbital motion. Similar trends are visible for $K = 9$. The maximum gap in request hit rate between LRU and StarCDN is 15% (for cache size=60 GB and $K = 9$).

Similar trends are visible in terms of the byte hit rate plotted in Fig. 7b and Fig. 7d. Byte hit rate captures the percentage of bytes being served from the cache. A higher byte hit rate implies a lower bandwidth utilization between ground and space. StarCDN can significantly reduce the bandwidth utilization compared to a simple LRU cache.

Finally, we plot the uplink bandwidth utilized by different schemes in Fig. 8. This is normalized to the current state-of-the-art, where all the data needs to be fetched from the ground. LRU uses 30-35% of the bandwidth compared to fetching everything from ground, which already demonstrates the benefits of caching in space. StarCDN can improve it even further and uses just 20-25% of the uplink bandwidth, demonstrating significant savings for LSN operators.

5.2.1 Benefits of Consistent hashing We evaluate how consistent hashing affects the request hit rate and byte hit rate when serving request traffic. Compared to Vanilla LRU, StarCDN-Fetch (i.e., StarCDN without fetch) results in an average increase of 6 points in request hit rate and an average of 4.8 points in byte hit rate, as shown in Fig. 7a and Fig. 7b. Fig. 7c and Fig. 7d illustrate that when we have $K = 9$ hash buckets, the average improvement of request hit rate (byte hit rate) grows by 9.7 points (7.8 points). A simple impact of doing inter-satellite hashing and satellite cache partitioning is to increase the effective cache size in space. However, the benefit of StarCDN's consistent hashing is beyond improved cache size. Consider Fig. 7c, where we have $K = 9$ hash buckets. The effective cache size increases by nine times. However, the request hit rate of the LRU baseline at 90GB (61.3%) is 3.1 points smaller than StarCDN hashing at 10GB (64.4%). This is because adjacent users could be scheduled to different satellites; therefore, identical requests might be processed in different caches if we are running



(a) *Request hit rate ($K = 4$)* **(b)** *Byte hit rate ($K = 4$)* **(c)** *Request hit rate ($K = 9$)* **(d)** *Byte hit rate ($K = 9$)*

Figure 7: *Request hit rate curve and byte hit rate curve for StarCDN variants, the LRU baseline, and the ideal static cache.*

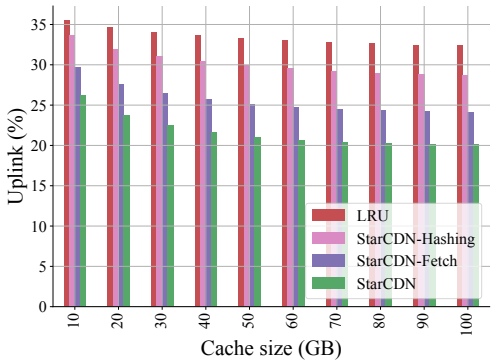


Figure 8: *Uplink usage of StarCDN variants ($K = 9$) schemes and the LRU baseline, normalized with respect to Starlink with no cache.*

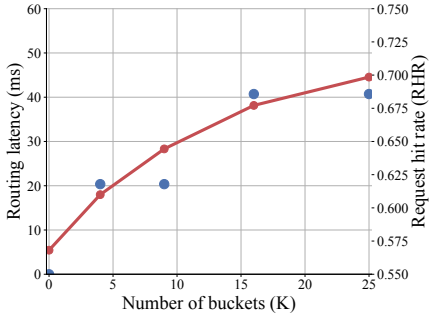


Figure 9: *Worst case routing latency to the satellite hosting the correct hash bucket (points) and request hit rate (curve) with respect to number of buckets K. Both latency and hit rate increase with K.*

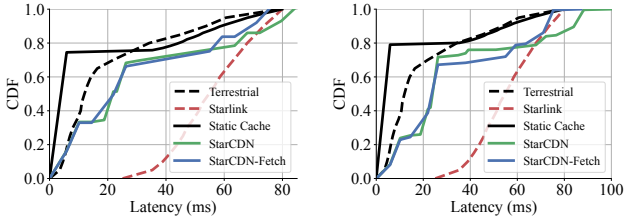
Vanilla LRU (e.g., User-1 and user-2 are scheduled to satellite D and B in Fig. 4). StarCDN addresses this problem by hashing and routing to ensure identical content will be served by the same cache (e.g., satellite D in Fig. 4), resulting in a higher hit rate.

5.2.2 Relayed fetch Relayed fetching between replicas leads to a 4.8 point (4.1 point) improvement in RHR and a 4.1(3.9) point improvement in BHR on top of StarCDN-Fetch with $K = 4$ ($K = 9$).

In §3.3, we describe how the relayed fetching utilizes the historical traffic footprint to boost performance. To evaluate this, we monitor the availability of a request in inter-orbital neighbors when a cache miss happens in Table. 3. In StarCDN, a request to the nearest inter-orbital neighboring replicas is sent when there is a cache

Cache size (GB)	West only		East only		Both	
	Req	Byte	Req	Byte	Req	Byte
10	47.5	47.5	31.4	28.5	11.9	8.86
50	61.6	62.3	30.1	29.3	14.6	13.6
100	64.7	64.9	27.4	26.5	14.7	13.8

Table 3: *Number of requests(Mil) and bytes(TB) available in inter-orbital neighbors with four hashing buckets during a miss*



(a) *Latency CDF of StarCDN variants with $K = 4$ hash buckets, Terrestrial CDN, regular Starlink (no cache), and the Static Cache.*

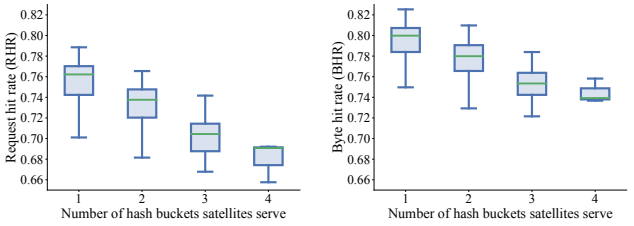
(b) *Latency CDF of StarCDN variants with $K = 9$ hash buckets, Terrestrial CDN, regular Starlink (no cache), and Static Cache.*

Figure 10: *Latency performance of StarCDN.*

miss. If the content is available in neighbors, it will be fetched and served from space. As the cache size grows, more and more requests can be fulfilled by the left neighbor alone instead of the right or both neighbors. This is because the left neighbor has recently visited the same location. A large fraction of this benefit comes from the fact that the satellite is able to access a preceding satellite's cache.

5.3 Latency

To evaluate the latency of StarCDN, we estimate the end-to-end propagation delay between a user initiating a request and that request being fulfilled either from space or terrestrial CDN. To the best of our knowledge, Starlink does not reveal any queueing delay and processing delay in their satellites. The baselines to which we are comparing are the terrestrial CDN and regular Starlink latency reported by [8] based on data analysis from the Cloudflare AIM dataset. The latency in the baseline is *idle* latency, which means the impact of queueing delay is minimized in the dataset, making our latency comparable. We plot the latency CDF in Fig. 10. The median latency of StarCDN is 22ms as compared to 55ms in regular Starlink with no cache bringing us closer to the performance of terrestrial CDNs (serving terrestrial users). Like terrestrial and Starlink CDN access, StarCDN has a long tail latency due to cache misses. When



(a) *Request hit rate* **(b)** *Byte hit rate*

Figure 11: *Hit rate performance of satellites serving different numbers of hash buckets.*

a cache miss occurs in StarCDN, all routings using ISLs become an overhead, and requests must be downlinked to terrestrial networks.

There is a tradeoff between the number of hash buckets and latency. Due to the limitation of grid-shaped ISLs, the more buckets we have, the more routing overhead is required to route requests to the correct hash buckets. Fig. 9 illustrates the worst routing latency for consistent hashing with no failure and the request hit rate when a 10 GB cache is deployed. As K grows beyond 9, the worst-case latency overhead becomes unaffordable (40ms) for routing even if it yields an additional hit rate of upto 5%. When we have $K = 9$ buckets, the routing overhead for consistent hashing remains the same as when we have $K = 4$ buckets ($2\lfloor \frac{\sqrt{K}}{2} \rfloor$ is the same for both configurations). This is reflected in StarCDN-Fetch in Fig. 10a and 10b where $K = 9$ has strictly better latency than $K = 4$. However, $K = 9$ can incur additional overhead for relayed fetching due to more ISLs required to reach a replica, which explains the worse tail latencies.

5.4 Fault Tolerance

We evaluate the robustness against unavailability through real-world satellite constellation status. When we collect the orbital shell information, we identify 126 out of 1296 (9.7%) active satellites that are out of slots. These unavailable satellites result in 438 broken ISLs among available satellites. Recall that StarCDN hashes every request to the correct bucket. Without unavailability, each satellite will serve one hash bucket. However, as ISLs become unavailable, some satellites will be remapped to serve multiple hash buckets due to StarCDN's consistent hashing scheme. We experimented StarCDN with nine hash buckets equipped with 50GB cache. We plot the hit rate of satellites serving different numbers of hashing IDs in Fig. 11. When a satellite serves more hashing IDs due to unavailability, the request hit rate (byte hit rate) drops up to 7 percentage points (5 percentage points). While this is a noticeable drop, StarCDN still saves 74% of uplink bandwidth in comparison with not using StarCDN.

5.5 Evaluation of Web and Download Delivery

The prior sections focus on video content delivery. Video content accounts for a large fraction of the bits served by a CDN. However, CDNs serve a wide range of traffic classes [58], such as websites and software downloads. Different traffic classes have different user access patterns, and object size distributions, and hence different caching properties. Therefore, it is important to evaluate and understand the performance of StarCDN for traffic classes other than videos, such as web and download traffic.

In this section, we focus on request hit rate and byte hit rate as they are crucial metrics directly correlated with uplink usage and user-perceived latency. Using SpaceGEN, we generate 5-day-long synthetic traces from Akamai's production traces. Our web synthetic traces have 2 billion requests for a total of 642TB of content, and download traces have 472 million requests for a total of 372TB of content. Compared to our video traces, the total amount of traffic and content footprint for both web and downloads is much lower. Consequently, the hit rate curves increase more gradually with cache size.

We illustrate StarCDN's hit rate performance in Fig. 12. For both content types, StarCDN outperforms the LRU baseline noticeably. Particularly in Fig. 12d, StarCDN boosts the byte hit rate for more than 30%. Further, as expected, the Static Cache provides an ideal upper bound of the cache hit rates, and fewer cache buckets ($K = 4$) provide less hit rate than more cache buckets ($K = 9$).

6 Related Work

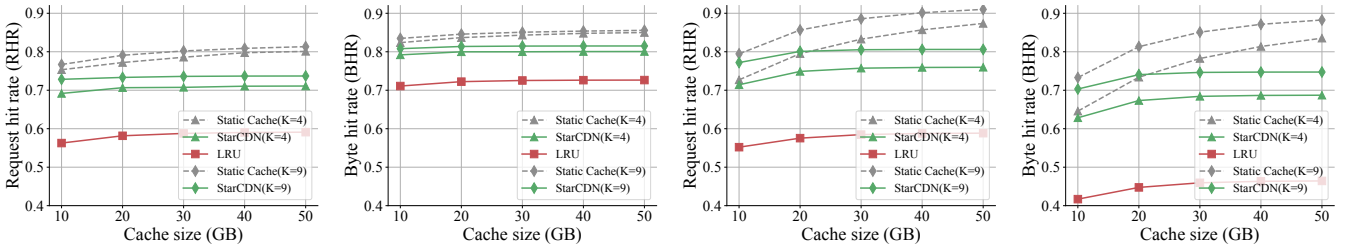
6.1 LEO Satellite Networks

LSNs have been greatly studied in the last decade with the emergence of services like Starlink, Oneweb and Kuiper. These networks are inherently dynamic and bandwidth-constrained due to satellite speeds [65, 68], frequent satellite handovers, and weather-induced variations [33]. Consequently, many works [28, 61, 62, 66] have measured and studied performance optimization on different layers of the networking stack.

Our work focuses on the broader domain of in-orbit computing. Past work, such as [7, 19, 64], has first proposed the idea of in-orbit computing in satellite networks for multiple applications such as CDNs, analysis of satellite imagery, etc. These designs leverage the emerging storage and compute capabilities in small LEO satellites. Recently, [8, 27] conducted extensive measurements to identify the shortcomings of terrestrial CDN providers in the context of LSNs (discussed in §1). Our work goes deeper into CDN architecture and design than past work in this space. We collect real-world traffic traces from multiple locations on the globe. Unlike past work, we propose a specific architecture for satellite-based CDNs, design new techniques, and evaluate these techniques using real-world traffic traces. In addition, we design a synthetic traffic generator that can be used to build new satellite-based caching systems.

6.2 CDNs and Caching Policies

There is extensive work in caching algorithms [3, 4, 6, 14, 31, 75] which make caching decisions based on various factors like popularity, object size, geography, time, cost, etc. [6, 13, 46, 57] to improve cache performance [5, 49, 58]. CDN's infrastructure has also been extensively studied. Leading companies like Cloudflare, Google, and Akamai publish their infrastructure designs in detail[17, 23, 24, 36]. Our work is orthogonal to these lines of work. We focus on the architecture of CDNs in LSNs and design new techniques to counter orbital motion, which is unique to LSNs. We choose LRU as our eviction algorithm of choice due to its simplicity and use in practical deployments. Besides eviction policies, we focus on key aspects of



(a) *Request hit rate for web traffic* **(b)** *Byte hit rate for web traffic* **(c)** *Request hit rate for downloads traffic* **(d)** *Byte hit rate for downloads traffic*

Figure 12: *Request hit rate curve and byte hit rate curve for web and downloads traffic*

content fetching that are uniquely helpful in an LSN context. While our work on synthetic trace generation is inspired by previous work [44, 45], we are the first to model the geographic diversity across multiple locations in synthetic traces, as described in §4.

7 Limitations and Future Work

In this section, we discuss some limitations in our simulation frameworks and trace generator, and future work in trace generation and satellite-based CDNs.

Cross-location Temporal Traffic Correlation: One limitation of our trace generator and traffic model is that it does not fully capture some subtle temporal correlations that are possible between different locations. For instance, a hot news item on CNN may become popular a few hours later in California than in Boston, due to the time difference. However, as we showed earlier, the hit rate gap between production and synthetic traces for satellite-based caches is very small, providing evidence that these correlations have a limited effect on cache performance.

Discrete-time Simulator: Due to the inherent limitations of the discrete-time simulator, our current simulation framework does not model disconnections during object transfer. When we replay the cache trace at each satellite, it will only yield a cache hit or a cache miss. A Starlink satellite triggers a handover every few minutes, thus incurs a potential transmission failure. Capturing this kind of behavior requires a complicated simulator. We left a more fine-grained LEO satellite simulator with link-layer simulation capability and the impact on critical transmission failure as a future work direction.

Constellation-CDN Co-design: In designing StarCDN, we model the Starlink network as it is deployed, based on publicly available information. However, there can be an alternate design which jointly optimizes constellation design and caching strategies, e.g., by modifying ISL topologies dynamically or changing ground-satellite mappings in response to cache misses. We do not explore such designs because constellation design is governed by multiple factors, such as regulation, feasibility, launch costs, etc. Therefore, we opt for independent designs for constellation and caching.

New Applications: Orbital motion of LEO satellites creates challenges for multiple layers of the stack. With the emergence of direct-to-cell services, we envision that maintaining state for users in a geographic region, as the underlying containers of the data move, will emerge as a challenge for satellite-based cell services. We believe this is an important area of future work and will require even more stringent latency guarantees.

Security and DNS: TLS and relevant encryption technologies are crucial for modern network applications. As described in [42], edge servers host cryptographic keys used for terminating TLS connections with a key management infrastructure (KMI). Similarly, a fast, efficient DNS infrastructure to resolve a client to the first-contact satellite also plays a vital role in the actual deployment of the system. We believe these are important topics for future work.

Co-optimizing CDNs and LSNs: An intermediate design between today's CDNs and StarCDN could be to place edge servers co-located with ground stations. While this design can be implemented today and improve QoE for users, it may not significantly reduce ground-satellite network utilization and user-perceived latency. However, jointly optimizing the traffic routing and content caching decisions of LSNs and terrestrial CDNs is worth exploring from both a performance and cost perspective.

8 Conclusion

We present StarCDN, a novel satellite-based CDN architecture that enables content to be cached efficiently in edge servers deployed in space. StarCDN implements mechanisms for placing and fetching content in satellite-based caches to reduce user-perceived latency and optimize the utilization of ground-to-satellite links. We also develop the first open-source trace generator, SpaceGEN, that can generate realistic synthetic traces of content access from a globally distributed set of users. We hope this tool and our dataset can promote future satellite-based CDN and caching research.

Ethical concerns – Our work uses anonymized production CDN traces and raises no ethical concerns.

Acknowledgment

We thank our shepherd and the anonymous reviewers for their valuable feedback. The work in this paper was partially funded by NSF Grants # 2237474, # 2106463, and # 1901137. We are grateful to the members of the wireless networking group at UIUC for feedback on early drafts.

References

[1] Vijay Kumar Adhikari, Sourabh Jain, Yingying Chen, and Zhi-Li Zhang. 2012. Vivisecting youtube: An active measurement study (INFOCOM 2012).
[2] Amazon. 2025. Project Kuiper. https://www.aboutamazon.com/what-we-do/devices-services/project-kuiper.

[3] Ismail Ari, Ahmed Amer, Robert Gramacy, Ethan L. Miller, Scott A. Brandt, and Darrell D. E. Long. 2002. ACME: Adaptive Caching Using Multiple Experts. Proceedings in Informatics (2002).

[4] Nathan Beckmann, Haoxian Chen, and Asaf Cidon. 2018. LHD: Improving Cache Hit Rate by Maximizing Hit Density (NSDI 2018).

[5] Daniel S. Berger. 2018. Towards Lightweight and Robust Machine Learning for CDN Caching (HotNets '18). Association for Computing Machinery.

[6] Daniel S. Berger, Ramesh K. Sitaraman, and Mor Harchol-Balter. 2017. AdaptSize: Orchestrating the Hot Object Memory Cache in a Content Delivery Network (NSDI 2017).

[7] Debopam Bhattacherjee, Simon Kassing, Melissa Licciardello, and Ankit Singla. 2020. In-orbit Computing: An Outlandish Thought Experiment? (HotNets 2020).

[8] Rohan Bose, Saeed Fadaei, Nitinder Mohan, Mohamed Kassem, Nishanth Sastry, and Jörg Ott. 2024. It's a bird? It's a plane? It's CDN!: Investigating Content Delivery Networks in the LEO Satellite Networks Era (HotNets 2024).

[9] Rohan Bose, Nitinder Mohan, and Jörg Ott. 2024. Poster: Twinkle, Twinkle, Streaming Star: Illuminating CDN Performance over Starlink (IMC 2024).

[10] Business Insider. 2024. Elon Musk's Starlink Satellites and Internet Services. https://www.businessinsider.com/elon-musk-starlink-satellites-internet. (2024).

[11] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. 2015. Analyzing the Performance of an Anycast CDN (IMC 2015).

[12] Celestrak. 2025. celestrak.org. https://celestrak.org/.

[13] Fangfei Chen, Ramesh K. Sitaraman, and Marcelo Torres. 2015. End-User Mapping: Next Generation Request Routing for Content Delivery (SIGCOMM 2015).

[14] Jiayi Chen, Nihal Sharma, Tarannum Khan, Shu Liu, Brian Chang, Aditya Akella, Sanjay Shakkottai, and Ramesh K Sitaraman. 2023. Darwin: Flexible Learning-based CDN Caching (SIGCOMM 2023).

[15] CISCO. 2011. White Paper: The future of networking for high-throughput data centers. https://twiki.cern.ch/twiki/pub/HEPIX/TechwatchNetwork/HtwNetworkDocuments/white-paper-c11-741490.pdf.

[16] Citizen Digital. 2023. Starlink Pauses New Subscriptions in Nairobi, Cites Network Overload. https://www.citizen.digital/tech/starlink-pauses-new-subscriptions-in-nairobi-cites-network-overload-n352395?utm_medium=Social&utm_source=Twitter#Echobox=1730728917

[17] Cloudflare. 2025. Content Delivery Network (CDN) Reference Architecture. https://developers.cloudflare.com/reference-architecture/architectures/cdn/

[18] Cloudflare. 2025. Starlink AS geographical traffic distribution. https://radar.cloudflare.com/traffic/as14593.

[19] Bradley Denby and Brandon Lucia. 2020. Orbital Edge Computing: Nanosatellite Constellations as a New Class of Computer System (ASPLOS 2020).

[20] John Dilley, Bruce M. Maggs, Jay Parikh, Harald Prokop, Ramesh K. Sitaraman, and William E. Weihl. 2002. Globally Distributed Content Delivery. IEEE Internet Comput. (2002).

[21] Marwan Fayed, Lorenz Bauer, Vasileios Giotsas, Sami Kerola, Marek Majkowski, Pavel Odintsov, Jakub Sitnicki, Taejoong Chung, Dave Levin, Alan Mislove, et al. 2021. The Ties that un-Bind: Decoupling IP from web services and sockets for robust addressing agility at CDN-scale (SIGCOMM 2021).

[22] Gianluca Giuffrida, Luca Fanucci, Gabriele Meoni, Matej Batič, Léonie Buckley, Aubrey Dunne, Chris van Dijk, Marco Esposito, John Hefele, Nathan Vercruyssen, Gianluca Furano, Massimiliano Pastena, and Josef Aschbacher. 2022. The -Sat-1 Mission: The First On-Board Deep Neural Network Demonstrator for Satellite Earth Observation. IEEE Transactions on Geoscience and Remote Sensing (2022).

[23] Google. 2025. Cloud CDN overview. https://cloud.google.com/cdn/docs/overview

[24] Google. 2025. Media CDN overview. https://cloud.google.com/media-cdn/docs/overview

[25] Syed Hasan, Sergey Gorinsky, Constantine Dovrolis, and Ramesh K. Sitaraman. 2014. Trade-offs in optimizing the cache deployments of CDNs (INFOCOM 2014).

[26] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A buffer-based approach to rate adaptation: Evidence from a large video streaming service (SIGCOMM 2014).

[27] Liz Izhikevich, Reese Enghardt, Te-Yuan Huang, and Renata Teixeira. 2025. A Global Perspective on the Past, Present, and Future of Video Streaming over Starlink (SIGMETRICS 2025).

[28] Liz Izhikevich, Manda Tran, Katherine Izhikevich, Gautam Akiwate, and Zakir Durumeric. 2024. Democratizing LEO Satellite Network Measurement. Proc. ACM Meas. Anal. Comput. Syst. (2024).

[29] Tommy R Jensen and Bjarne Toft. 2011. Graph coloring problems. John Wiley & Sons.

[30] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. 1997. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web (STOC 1997).

[31] Vadim Kirilin, Aditya Sundarrajan, Sergey Gorinsky, and Ramesh K. Sitaraman. 2019. RL-Cache: Learning-Based Cache Admission for Content Delivery (NetAI@SIGCOMM 2019).

[32] Planet Labs. 2024. Planet Launches High-Resolution Pelican-2 Satellite, 36 SuperDoves. https://www.planet.com/pulse/planet-launches-high-resolution-pelican-2-satellite-36-superdoves/

[33] Dominic Laniewski, Eric Lanfer, Bernd Meijerink, Roland van Rijswijk-Deij, and Nils Aschenbruck. 2024. WetLinks: A Large-Scale Longitudinal Starlink Dataset with Contiguous Weather Data (TMA 2024).

[34] James Larisch, Timothy Alberdingk Thijm, Suleman Ahmad, Peter Wu, Tom Arnfeld, and Marwan Fayed. 2024. Topaz: Declarative and verifiable authoritative DNS at CDN-scale (SIGCOMM 2024).

[35] Yuanjie Li, Hewu Li, Wei Liu, Lixin Liu, Wei Zhao, Yimei Chen, Jianping Wu, Qian Wu, Jun Liu, Zeqi Lai, and Han Qiu. 2023. A Networking Perspective on Starlink's Self-Driving LEO Mega-Constellation (MobiCom 2023).

[36] Bruce M Maggs and Ramesh K Sitaraman. 2015. Algorithmic nuggets in content delivery. ACM SIGCOMM Computer Communication Review (2015).

[37] Michael Kan. 2021. Starlink's Laser System Is Beaming 42 Million GB of Data Per Day. https://www.pcmag.com/news/starlinks-laser-system-is-beaming-42-million-gb-of-data-per-day.

[38] Microsoft. 2025. CosmicBeats-Simulator. https://github.com/microsoft/CosmicBeats-Simulator

[39] Mike Puchol. 2025. starlink.sx. https://starlink.sx/.

[40] Nitinder Mohan, Andrew E. Ferguson, Hendrik Cech, Rohan Bose, Prakita Rayyan Renatin, Mahesh K. Marina, and Jörg Ott. 2024. A Multifaceted Look at Starlink Performance (WWW 2024).

[41] NASA. 2023. High-Performance Spaceflight Computing (HPSC). https://www.nasa.gov/game-changing-development-projects/high-performance-spaceflight-computing-hpsc/

[42] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. 2010. The Akamai network: a platform for high-performance internet applications. ACM SIGOPS Oper. Syst. Rev. (2010).

[43] Oneweb. 2025. Oneweb. https://oneweb.net/.

[44] Anirudh Sabnis and Ramesh K. Sitaraman. 2021. TRAGEN: A Synthetic Trace Generator for Realistic Cache Simulations (IMC 2021).

[45] Anirudh Sabnis and Ramesh K. Sitaraman. 2022. JEDI: Model-driven trace generation for cache simulations (IMC 2022).

[46] Fethi Sazoglu, Berkant Cambazoglu, Rifat Ozcan, Ismail Altingövde, and Özgür Ulusoy. 2013. A financial cost metric for result caching (SIGIR 2013).

[47] Kyle Schomp, Onkar Bhardwaj, Eymen Kurdoglu, Mashooq Muhaimen, and Ramesh K. Sitaraman. 2020. Akamai DNS: Providing Authoritative Answers to the World's Queries (SIGCOMM 2020).

[48] Jayanth Shenoy, Om Chabra, Tusher Chakraborty, Suraj Jog, Deepak Vasisth, and Ranveer Chandra. 2024. CosMAC: Constellation-Aware Medium Access and Scheduling for IoT Satellites (MobiCom 2024).

[49] Zhenyu Song, Daniel S. Berger, Kai Li, and Wyatt Lloyd. 2020. Learning Relaxed Belady for Content Distribution Network Caching (NSDI 2020).

[50] SpaceX. 2022. Tweet from SpaceX 1M users. https://x.com/SpaceX/status/1604872936976154624

[51] Starlink. 2021. Petition of Starlink Services, LLC For Designation as an Eligible Telecommunications Carrier. https://pdfhost.io/v/BnYWSR~wq_Starlink_Services_LLC_Application_for_ETC_Designation.

[52] Starlink. 2024. Direct to Cell. https://www.starlink.com/business/direct-to-cell.

[53] Starlink. 2024. Starlink coverage expanded to new regions! https://x.com/Starlink/status/1839424733198344617.

[54] Starlink. 2025. Starlink. https://www.starlink.com/us.

[55] Starlink. 2025. Starlink Technology. https://www.starlink.com/us/technology.

[56] StarlinkUsers. 2025. Tweet from Starlink 4M users. https://x.com/Starlink/status/1839424733198344617

[57] Kalika Suksomboon, Saran Tarnoi, Yusheng Ji, Michihiro Koibuchi, Kensuke Fukuda, Shunji Abe, Nakamura Motonori, Michihiro Aoki, Shigeo Urushidani, and Shigeki Yamada. 2013. PopCache: Cache more or less based on content popularity for information-centric networking (IEEE Conference on Local Computer Networks 2013).

[58] Aditya Sundarrajan, Mingdong Feng, Mangesh Kasbekar, and Ramesh K Sitaraman. 2017. Footprint Descriptors: Theory and Practice of Cache Provisioning in a Global CDN (CoNEXT 2018).

[59] Aditya Sundarrajan, Mangesh Kasbekar, Ramesh K. Sitaraman, and Samta Shukla. 2020. Midgress-aware traffic provisioning for content delivery (ATC 2020).

[60] Mercury Systems. 2023. Mercury Systems Ships First Space-Qualified Commercial Solid-State Data Recorder. https://ir.mrcy.com/news-releases/news-release-details/mercury-systems-ships-first-space-qualified-commercial-solid

[61] Aryan Taneja, Debopam Bhattacherjee, Saikat Guha, and Venkata N. Padmanabhan. 2023. On viewing SpaceX Starlink through the Social Media Lens. https://arxiv.org/abs/2307.13441.

[62] Aryan Taneja, Rahul Bothra, Debopam Bhattacherjee, Rohan Gandhi, Venkata N. Padmanabhan, Ranjita Bhagwan, Nagarajan Natarajan, Saikat Guha, and Ross Cutler. 2023. Don't Forget the User: It's Time to Rethink Network Measurements (HotNets 2023).

[63] Qingqing Tang, Zesong Fei, Bin Li, and Zhu Han. 2021. Computation Offloading in LEO Satellite Networks With Hybrid Cloud and Edge Computing. IEEE Internet of Things Journal (2021).

[64] Bill Tao, Om Chabra, Ishani Janveja, Indranil Gupta, and Deepak Vasisht. 2024. Known Knowns and Unknowns: Near-realtime Earth Observation Via Query Bifurcation in Serval (NSDI 2024).
[65] Bill Tao, Maleeha Masood, Indranil Gupta, and Deepak Vasisht. 2023. Transmitting, Fast and Slow: Scheduling Satellite Traffic through Space and Time (MobiCom 2023).
[66] Shubham Tiwari, Saksham Bhushan, Aryan Taneja, Mohamed Kassem, Cheng Luo, Cong Zhou, Zhiyuan He, Aravindh Raman, Nishanth Sastry, Lili Qiu, and Debopam Bhattacherjee. 2023. T3P: Demystifying Low-Earth Orbit Satellite Broadband. https://arxiv.org/abs/2310.11835.
[67] Ruben Torres, Alessandro Finamore, Jin Ryong Kim, Marco Mellia, Maurizio M Munafo, and Sanjay Rao. 2011. Dissecting video server selection strategies in the YouTube CDN (ICDCS 2011).
[68] Deepak Vasisht, Jayanth Shenoy, and Ranveer Chandra. 2021. L2D2: low latency distributed downlink for LEO satellites (SIGCOMM 2021).
[69] Mike Wall. 2024. AI in Space: NVIDIA GPU Heads to Orbit on SpaceX's Transporter-11 Mission. https://www.space.com/ai-nvidia-gpu-spacex-launch-transporter-11
[70] Ruolin Xing, Mengwei Xu, Ao Zhou, Qing Li, Yiran Zhang, Feng Qian, and Shangguang Wang. 2024. Deciphering the Enigma of Satellite Computing with COTS Devices: Measurement and Analysis (MobiCom 2024).
[71] Juncheng Yang, Anirudh Sabnis, Daniel S. Berger, K. V. Rashmi, and Ramesh K. Sitaraman. 2022. C2DN: How to Harness Erasure Codes at the Edge for Efficient Content Delivery (NSDI 2022).
[72] Li Yuanjie, Liu Lixin, Li Hewu, Liu Wei, Chen Yimei, Zhao Wei, Wu Jianping, Wu Qian, Liu Jun, and Lai Zeqi. 2024. Stable Hierarchical Routing for Operational LEO Networks (MobiCom 2024).
[73] Xiao Zhang, Tanmoy Sen, Zheyuan Zhang, Tim April, Balakrishnan Chandrasekaran, David Choffnes, Bruce M Maggs, Haiying Shen, Ramesh K Sitaraman, and Xiaowei Yang. 2021. Anyopt: Predicting and optimizing ip anycast performance (SIGCOMM 2021).
[74] Yazhuo Zhang, Juncheng Yang, Yao Yue, Ymir Vigfusson, and K.V. Rashmi. 2024. SIEVE is Simpler than LRU: an Efficient Turn-Key Eviction Algorithm for Web Caches (NSDI 2024).
[75] Behrouz Zolfaghari, Gautam Srivastava, Swapnoneel Roy, Hamid R. Nemati, Fatemeh Afghah, Takeshi Koshiba, Abolfazl Razi, Khodakhast Bibak, Pinaki Mitra, and Brijesh Kumar Rai. 2020. Content Delivery Networks: State of the Art, Trends, and Future Roadmap. ACM Comput. Surv. (2020).

A Appendix

Appendices are supporting material that has not been peer-reviewed.

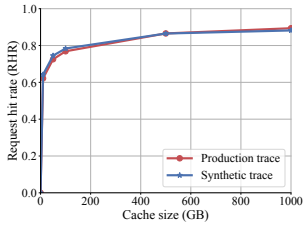
A.1 Algorithm of trace generator

Algorithm 1: Correlated Trace Generation

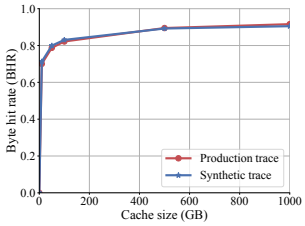
Input: n pFDs = $\{\langle\lambda_1, P_1^r, P_1^a\rangle, \ldots, \langle\lambda_n, P_n^r, P_n^a\rangle\}$, correlation distribution $P^c(p_1, \ldots, p_n, z)$, trace length N;
Output: n synthetic traces $\{\Pi_i\}$;
Phase 1: Initializing
Compute $P_i^r(s|p, z)$ $\forall$ pFDs;
Initialize empty lists C_i $\forall$ pFDs;
Compute largest finite stack distance C_i^{max} $\forall$ pFD ;
request_cnt = {}, objID = 0;
while $any(C_i.size < C_i^{max})$ **do**
  Sample $p_1, \ldots, p_n, z$ from P^c;
  forall $p_i \neq 0$ **do**
    Add (objID, p_i, z) to C_i;
    C_i.size += z;
  end
  objID += 1;
end
Phase 2: Synthetic Generation
$\Pi_i = \emptyset$ for all i, $iter = 0$;
Initialize reqRateCounter[i] to 0, load reqRate[i] $\forall$ pFD ;
while counter < N **do**
  forall i **do**
    if reqRateCounter[i] > 0 **then**
      Pop first object $\langle o_{id}, o_p, o_z\rangle$ in C_i and append to Π_i ;
      $request\_cnt[i][o_{id}]$ += 1;
      if $o_p = request\_cnt[i][o_{id}]$ **then**
        Sample new objects like initialization phase;
      else
        Sample stack distance s from $P^r(s|o_p, o_z)$;
        Insert o to location j in C_i such that $\sum_{k<j} o_{k_z} \geq s$;
      end
      reqRateCounter[i] −= 1;
    end
    reqRateCounter[i] += reqRate[i];
  end
  counter += 1;
end
Assign timestamps to the traces;
return $\{\Pi_i\}$;

A.2 Properties of Synthetic Traces

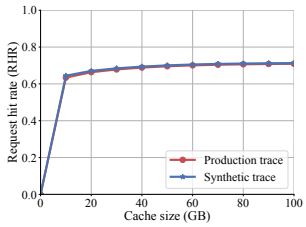
In this section, we show results from simulations of the StarCDN-Fetch architecture using synthetic and production traces.



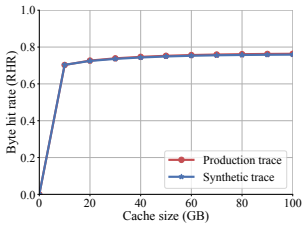
(a) *Request hit rate of two traces in terrestrial cache emulation.*



(b) *Byte hit rate of two traces in terrestrial cache emulation.*



(c) *Request hit rate of two traces in StarCDN-Fetch emulation.*



(d) *Byte hit rate of two traces in StarCDN-Fetch emulation.*

Figure 13: *Hit rate comparison of production and synthetic traces in evaluation benchmarks in §5.*

Fig. 13c and Fig. 13d show that difference in the measured cache hit rates between the two traces is small. This provides further evidence that the synthetic traces can be used in lieu of the production traces for our evaluation.