Cooperatively Constructing Cost-Effective Content Distribution Networks upon Emerging Low Earth Orbit Satellites and Clouds

Zeqi Lai, Hewu Li*, Qi Zhang, Qian Wu, Jianping Wu
Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University, Beijing 100084
zeqilai@tsinghua.edu.cn, {lihewu, wuqian, jianping}@cernet.edu.cn, qi-zhang19@mails.tsinghua.edu.cn

***Abstract*—Internet content providers typically exploit cloud-based content delivery/distribution networks (CDNs) to provide wide-area data access with high availability and low latency. However, from a global perspective, a large portion of users still suffer from high content access latency due to the insufficient deployment of terrestrial cloud infrastructures.**

This paper presents StarFront, a cost-effective content distribution framework to optimize global CDNs and enable low content access latency *anywhere*. StarFront builds CDNs upon emerging low Earth orbit (LEO) constellations and existing cloud platforms to satisfy the low-latency requirements while minimizing the operational cost. Specifically, StarFront exploits a key insight that emerging mega-constellations will consist of thousands of LEO satellites equipped with high-speed data links and storage, and thus can *potentially* work as *"cache in space"* to enable *pervasive* and *low-latency* data access. StarFront judiciously places replicas on either LEO satellites or clouds, and dynamically assigns user requests to proper cache servers based on constellation parameters, cloud/user distributions and pricing policies. Extensive trace-driven evaluations covering geo-distributed vantage points have demonstrated that: StarFront can effectively reduce the global content access latency with acceptable operational cost under representative CDN traffic.

***Index Terms*—Geo-distributed Content Distribution, LEO Constellations, Integrated Satellite and Terrestrial Networks.**

I. Introduction

Content distribution networks (CDNs) consist of a considerable number of geo-distributed cloud-based cache servers, with the goal of providing high availability and low content access latency globally. CDNs are carrying a significant amount of network traffic. The global CDN is expected to deliver about 72% of total Internet traffic by 2022 [5]. Therefore, optimizing the network performance of CDNs can significantly improve the quality of experience (QoE) of a variety of applications built upon it (*e.g.,* Web services and Video-on-Demand, *etc.*).

One of the key benefits enabled by CDNs is the low *content access latency*, which is typically quantified by the time consumption of delivering requested objects to end users. For example, the content access latency can refer to the page load time or the video initialization time for Web or VoD applications respectively. Critical to the low-latency story is that users can access content replicas cached on geo-distributed cache servers *close* to end users. However, our in-depth analysis on a large-scale CDN trace collected from seven major CDN operators across 183 countries reveals that a large portion of users are still suffering from high round-trip time (RTT) to their closest cache server, which can probably result in long content access latency. Our further analysis identifies that such high access latency is more prevalent in remote or rural areas, due to the insufficient deployment of cloud infrastructures as well as meandering terrestrial routes to cloud servers (*e.g.,* prolonged Internet paths caused by remote peering [28]).

However, none of the above root causes are easy to address in terrestrial networks. First, the deployment of today's Internet is essentially an *uneven* network, where network resources are aggregated in many developed regions (*e.g.,* "hot areas"). In remote and/or other under-developed areas, Internet access is limited, and provisioning and maintaining cloud servers to improve network performance is likely to be more expensive even though the population in such areas could be very large [19]. Second, terrestrial Internet is divided into many autonomous systems (ASes), and due to the specific routing policies of different ASes, inter-AS routes might be tortuous, resulting in meandering routes between users and assigned cloud servers which further increase the latency. *How can we handle the above challenges?*

Emerging mega-constellations with thousands of satellites flying in low Earth orbit (LEO) raise a new opportunity to optimize the network performance of CDNs globally. Modern satellites will be equipped with "high-throughput communication components" [34], [56], [60] and high-capacity storage [40], [42], and such satellites have the potential to be "cache in space". Intuitively, caching content replicas on emerging constellations is a promising approach to enable low latency pervasively [25]. However, fully leveraging the potential of mega-constellations needs to address several fundamental challenges: (i) satellites are fundamentally *mobile*, and moving at high-speed. How to properly select LEO satellites to cache content replicas and avoid the impact of intermittent connectivity? (ii) it is more expensive to carry traffic over satellites than over terrestrial networks. *How to judiciously assign user requests to a satellite or a cloud cache server in a cost-effective manner?*

We propose StarFront, a content distribution framework that cooperatively leverages cache nodes in both LEO satellites and terrestrial clouds to optimize the content access latency in a cost-effective manner. In particular, StarFront adopts two key techniques to achieve cost-effective wide-area content distribution. First, StarFront constructs a *dynamic satellite-cloud topology* which captures the time-varying accessibility

*Hewu Li is the corresponding author.

978-1-6654-4131-5/21/$31.00 ©2021 IEEE

and performance of the satellite-cloud integrated architecture. The construction process is based on the information of cloud distribution, predictable satellite trajectory together with their estimated performance and pricing models. Second, StarFront enables three forms of assignments for users in different regions. Specifically, a user request can be: (i) directly assigned to a cloud server (*i.e.,* a terrestrial cache) via terrestrial networks like in existing cloud-based CDNs; (ii) assigned to a cloud server through low-latency space routes constructed by a sequence of satellites; or (iii) assigned to a satellite cache server if the nearest cloud is still too far away. The above approaches of request assignment involve different latency performance and corresponding storage and traffic costs in practice. StarFront judiciously pushes contents and places replicas on available cloud or satellite servers, and assigns user requests to proper cache servers to meet the latency requirement of different applications, while minimizing the total cost of content distribution via satellite-cloud cooperation.

We have implemented the StarFront prototype based on Apache Traffic Server (ATS) [3]. We also build a testbed to simulate cloud sites and the dynamic LEO satellite network to evaluate the proposed framework. Trace-driven evaluations based on the state-of-the-art constellations (*e.g.,* Starlink and OneWeb) covering a collection of geo-distributed vantage points demonstrate the effectiveness of StarFront. By integrating satellites and clouds, StarFront outperforms existing cloud-only approaches and can satisfy various latency requirements from applications, at an acceptable operational cost. In addition, we find that the architectural design of constellation can significantly affect the performance of StarFront, as constellations with lower orbital altitude and equipped with inter-satellite links (ISLs) can further reduce more content access latency for terrestrial users.

Taken together, this paper makes three key contributions:

- We identify and analyze the high access latency problem in existing CDNs through a measurement study on seven commercial CDN operators (§II), and envision the feasibility and challenges of exploiting LEO mega-constellations to assist pervasive, low-latency content distribution (§III).
- We design StarFront, and build a prototype of it. StarFront is a content distribution framework that cooperatively leverages the storage and network capabilities in both clouds and LEO satellites to judiciously optimize the *content access latency* globally in a cost-effective manner (§IV).
- We evaluate the effectiveness of StarFront on improving global delivery efficiency via extensive trace-driven simulations on StarFront's prototype (§V).

II. Background and Motivation

Quick primer for wide-area content distribution. A content delivery/distribution network (CDN) is a highly-distributed platform with many cache servers separated globally. Content providers typically leverage public cloud platforms (*e.g.,* Amazon AWS, MS Azure, *etc.*) to deploy cache servers close to end users. Popular contents from the source server owned by the content provider are distributed to geographical cloud servers and in general the distribution process includes three key steps: (i) pushing the original contents from the source server to a collection of geo-distributed cache servers (*e.g.,* through building a distribution tree [29]); (ii) configuring the region↔server map which at runtime determines how user requests from different regions should be assigned to a specific cache server; and (iii) maintaining and updating data in each cache server, if there are new contents available from the source server. The above operations performed for content providers are charged primarily based on the storage and bandwidth they consumed, following the concrete pricing policies specified by different cloud providers (*e.g.,* [1], [2]).

Ideally, by moving contents close to users, wide-area CDNs are expected to geographically enable low *content access latency*, which is defined as the time consumption of delivering one or a batch of object(s) to end users. In practice, the content assess latency is mainly affected by the round-trip time (RTT) between users and the cache server they are assigned to, as well as the available bandwidth. The later typically depends on ISP or cloud provider pricing, while the former is constrained by the network topology.

High content access latency observed from a global perspective. To quantitatively understand the achievable performance of state-of-the-art wide-area commercial CDNs, we collect RTT measurements from 17402 probes across 183 countries to their nearest cloud sites provisioned by seven of the most popular CDN platforms (*i.e.,* Akamai, Azure, Cachefly, Cloudflare, Cloudfront, Fastly, and GoogleCloud). The measurements were conducted during the period between March-7 and April-1 2019, using the RIPE Atlas measurement platform [10]. For each probe we perform more than twenty `ping` tests to each of the seven CDNs considered, and gather the best probing results(access latency) to refrain the impact of network congestion or temporary link failure on the measurement results. We also use `traceroutes` tool to track how packets are routed between the users and cloud servers. Figure 1 plots the CDF of the RTT results from different geo-distributed users to the nearest cloud server of a certain CDN operator. As shown in Figure 1, *despite the advantages of CDNs (*e.g., *RTT less than 23ms for 40% of Akamai measurements), there is still a large fraction of CDN users suffering from RTT higher than 50ms, with a long tail of up to about 300ms, even if the closest cloud server is selected.*

High network RTTs can significantly impair the user-perceived experience, especially for time-sensitive applications with a batch of sequential requests (*e.g.,* Web browsing, video on-demand *etc.*). For example, as shown in [27], even tens of milliseconds of additional RTT might substantially deteriorate Web browsing page load times and degrade user experience.

To further understand the high RTT observations, we group the measurement result of each user by their original continents, as shown in Figure 2. We find that while the latency in many populated and developed areas is low, there are still a large number of users suffering from high access RTT, even if the nearest cloud server is selected. This is especially true for those users in remote or under-developed areas. Specifically, there

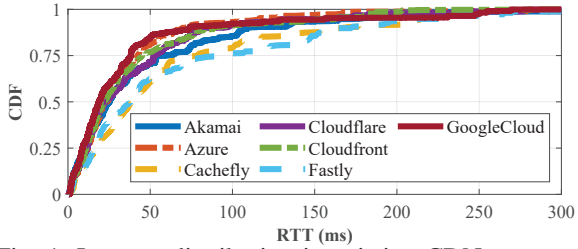

Fig. 1: Latency distribution in existing CDN operators.

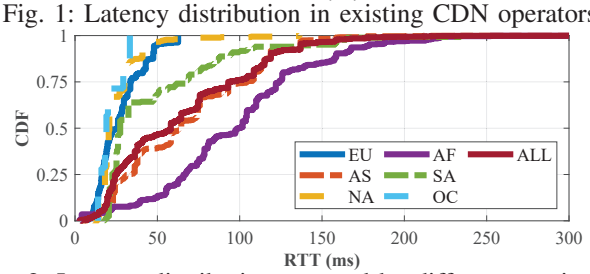

Fig. 2: Latency distribution grouped by different continents.

are about 53.58%/23.93% users suffering from RTT higher than 50ms/100ms in total. Even in developed regions like EU and NA, we still observe 3.4% and 4.5% users associated with RTT higher than 50ms respectively. The latency problem is more stringent in AF, where more than 86.6%/49.8% users suffer from RTT higher than 50ms/100ms.

Massively deploying cache servers in remote/rural regions is difficult. A straightforward approach to optimize the uneven CDN deployment and reduce access latency globally is to extend the coverage of terrestrial cloud sites to serve users in remote and rural areas, through massively increasing cloud/edge deployments in those regions. However, the terrestrial deployment is largely constrained by geographic factors (*e.g.,* terrain) or massive operation and maintenance cost, especially in under-developing areas. For example, in some regions, there may have no sufficient power infrastructure support for cloud/edge sites, or deployments in these areas are made difficult due to political or jurisdictional reasons. The *status quo* of today's terrestrial cloud-based CDNs motivates us to explore a challenging yet important question: *how can we address the above challenges and enable pervasive and low-latency CDN access globally?*

III. The Low-latency opportunity in Futuristic LEO mega-constellations

The research and deployment of low Earth orbit (LEO) satellites are re-gaining popularity in recent years [25], [26], [32], [35]–[38], [40], [44], [48], [54]. The rapid evolution of on-board technologies have improved the hardware capability on state-of-the-art LEO satellites. As compared to the first generation of satellite network that uses geostationary satellite for communication, emerging mega-constellations (*e.g.,* Starlink [15], OneWeb [8]) which plan to consist of thousands of mass-produced, low-flying satellites will be empowered with evolved network and storage capability, and thus (*likely*) enable new opportunities for constructing low-latency CDNs globally.

(i) Evolved on-board communication capability. Modern satellites are equipped with "high-throughput communication components" [34], [56], [60] which are able to provide tens or even hundreds of Gbps datarate [50]. Many planned constellations suggest the use of RF or laser inter-satellite links through which LEO satellite can connect to visible neighbor satellites and construct a network in space. In addition, due to its low-flying property (*i.e.,* 500-1200km altitude), LEO constellations also promise low-latency Internet connectivity, as compared to traditional geostationary satellites orbiting at about a 36,000km altitude suffering from high propagation delay. Another critical thing for realizing reduced latency is that the speed of light in terrestrial fiber is about 33% slower than that in air or vacuum [35]. Recent studies have outlined the vision of low-latency routing in space [36]–[38], [48], revealing the potential of reducing end-to-end latency via space routes over LEO spacecrafts, especially for long-distance communications.

(ii) Big data stores in space. Another evolution of on-board capacity is the storage in space. Recent works [40], [42] have envisioned the satellite-based big data storage. Cloud Constellation Corporation (CCC) built by SpaceBelt [13] is a data storage service using LEO satellites. The CCC system contains a ring of 10 LEO satellites in a 650-kilometer equatorial orbit, and three of them are data stores, offering about 5PB storage capacity since December 2018 [12].

The above evolutions from the space industry thus plot a promising picture of a satellite-cloud cooperative content distribution architecture that can (*potentially*) improve the accessibility and network performance of existing CDNs. Intuitively, LEO satellites can assist current cloud-based CDNs via: (i) constructing low-latency, close-to-optimal space paths connecting terrestrial clouds and end users to avoid meandering fiber/cable routes which might prolong the access latency; and (ii) enabling a new paradigm "*cache servers on LEO satellites*" that provides lower access latency for users in regions where even the nearest cloud site is still too far away. However, while promising, we argue that constructing such a cooperative CDN upon cloud data centers and mega-constellations still faces several unsolved challenges, due to two specific characteristics in LEO satellite networks.

(i) Scarce and costly space resources. While evolved, network and storage resources are still relatively limited and costly in space, as compared to the well-optimized terrestrial cloud platforms. For instance, transferring 1GB data traffic takes about $0.1 for the user of existing CDN operators (*e.g.,* Amazon CloudFront) [1]. As most emerging LEO constellations are still under heavy deployment and their pricing policies are not yet available, thus we estimate the cost of satellite data transfer based on current operating broadband satellite systems. For instance, transferring 1GB data volume takes about $1 by satellite networks, according to the pricing policy of ViaSAT [17], a satellite provider offering high-speed satellite-based broadband services. Moreover, for users of the satellite network, there is an additional upfront cost for the user terminal, *e.g.,* a small satellite dish [14]. As content providers typically have a cost budget for distributing their data over CDN, contents should be judiciously distributed to satellites and clouds in a cost-effective manner, *i.e.,* the constructed cooperative CDN is expected to satisfy the latency requirement of applications, while involving acceptable operating cost.

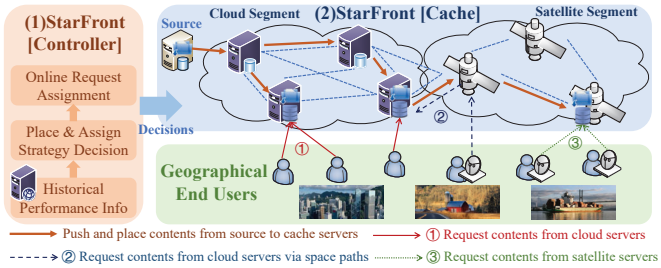

Fig. 3: The overview of STARFRONT framework.

(ii) High mobility of LEO satellites. LEO satellites are moving in high-speed with the respect to the Earth, resulting in unstable ground communication. Endhosts or cloud sites on the ground can only communicate to a LEO satellite if only it moves into the line of sight (LoS). In one orbit period (*e.g.,* which is about 90min for a Starlink Phase-I LEO satellite), one LEO satellite is only visible for a certain ground vantage point within several minutes. Application-level session might be interrupted if the request is assign to a satellite leaving the LoS. Therefore, user requests should be properly assigned to avoid performance degradation caused by intermittent connectivities.

IV. STARFRONT FRAMEWORK

To tackle the above challenges, we present STARFRONT, a cooperative content distribution framework that leverages a number of cache servers hosted either upon *LEO satellites* or globally distributed *cloud data centers* to construct pervasive low-latency CDNs in a cost-effective manner. At high-level, STARFRONT exploits the following key ideas: (i) leveraging the information of cloud distribution and predictable satellite trajectory, together with their estimated performance and pricing model to construct a *dynamic satellite-cloud topology* which captures the time-varying accessibility and performance of the satellite-cloud integrated architecture; and accordingly (ii) judiciously placing contents on the dynamic satellite-cloud topology, and assigning user requests to proper cache servers to meet the latency requirement of various applications, while minimizing the total cost of content distribution.

A. STARFRONT *System Overview*

Figure 3 plots the overview of STARFRONT, which consists of two core components: the STARFRONT geo-distributed cache and the STARFRONT controller. Specifically, the STARFRONT cache incorporates a *cloud segment* which is built upon geo-distributed cloud data centers (*i.e.,* terrestrial cache servers) offered by public cloud providers (*e.g.,* Amazon AWS), together with a *satellite segment* that is built upon emerging mega-constellations managed by satellite operators like SpaceX or OneWeb. Cloud data centers communicate with each other via terrestrial networks, while satellites equipped with inter-satellite links (ISLs) can transfer data to other satellites through satellite paths constructed by space routing algorithms (*e.g.,* [36]). Moreover, leading cloud providers are also actively deploying *ground-station-as-a-service (GSaaS)* [4], [21] which can enable *pay-as-you-go* ground communications to inter-connect cloud data centers and satellites. Leveraging GSaaS infrastructures, terrestrial clouds are able to communicate to a satellite on-demand if the satellite moves into the LoS.

Content providers can pick a collection of cache servers in STARFRONT framework and use them to distribute their contents to end users, as both cloud data centers and satellites with storage can be selected to cache contents. At runtime the STARFRONT framework allows content providers to execute two basic operations: (i) *content push and placement* operation, which refers to push contents from the source server to the set of selected cache servers via inter-cloud/satellite paths; and (ii) *request assignment* operation, which redirects user requests from geo-distributed regions to a proper cache server to obtain low access latency. Both operations involve fees according to the dedicated pricing policy of cloud and satellite operators, as the distribution process may consume storage and bandwidth resources in corresponding cloud or satellite platforms.

Collectively, STARFRONT enables three forms of request assignment with different network performance and corresponding costs. First, like traditional cloud-CDNs, STARFRONT allows users to request contents directly from a nearby cloud cache server (*e.g.,* ① in Figure 3), if the cloud performance can satisfy the application requirement. Second, if the access latency from a user to its closest cloud cache server is high (*e.g.,* due to the insufficient deployment of cloud sites, very-long communication distance or meandering terrestrial fiber route), STARFRONT allows users to accelerate the content access via free-space satellite routes (*e.g.,* ② in Figure 3). In this form, related satellites just forward traffic between users and the corresponding cloud cache servers, and do not need to cache replicas on the satellite. Third, requests can also be directly assigned to a satellite cache (*e.g.,* ③ in Figure 3) when the closest available cloud is still too far. Note that the above three forms of request assignment lead to different storage and bandwidth cost in practice, with resources in space being likely to be more precious. In particular, the first form only consumes terrestrial storage and bandwidth, the second form uses terrestrial storage and satellite bandwidth, while the last form consumes more expensive storage and bandwidth in space.

B. STARFRONT *Workflow*

To utilize the low-latency potential of emerging mega-constellations while not incurring high operating overhead for content providers, the STARFRONT controller judiciously calculates the placement and assignment decision in a cost-effective manner, *i.e.,* deciding *how to select a set of cache servers from all available clouds and satellites to form the content distribution network?* and *how to assign requests from different geo-distributed regions to a proper cache server while not exceeding the cost budget of content providers?* To deal with the high-dynamicity of constellation topology, the STARFRONT controller exploits the *periodicity* and *predictability* of satellite movement to periodically perform the following operations to accomplish the cost-effective distribution goal. Each operation will be described in details in following subsections.

- (i) At the beginning of each period, constructing a *dynamic satellite-cloud topology* to model the time-varying availability,

network performance, and cost of available clouds and satellites, based on cloud and user distributions, constellation pattern and pricing policies from cloud and satellite providers.
- (ii) Once the satellite-cloud topology is obtained, performing the *judicious placement and assignment* algorithm, and following the calculated decision to push and place contents on all selected cache servers, and then configuring the $region \leftrightarrow server$ assignment. In practice, the request assignment typically can be applied by local DNS redirection [59].
- (iii) During the period, redirecting user requests to a proper cache server, based on the $region \leftrightarrow server$ assignment. Intuitively, STARFRONT might assign requests to a nearby cloud cache server via terrestrial networks for users in developed areas (*e.g.,* metropolis) where low-latency cloud platforms are well-provisioned, and issue requests to a cloud server via a satellite path or directly to an available satellite cache server, for those users in remote or rural areas where terrestrial networks and clouds are quite limited.

C. Modeling the Time-varying Satellite-Cloud Topology

1) Formulating the dynamic satellite-cloud topology: While a number of previous works have studied the cost-effective content distribution problem in cloud platforms (*i.e.,* how to deploy replicas on statically-deployed cloud storage servers [29], [47], [58]), the high-dynamicity of LEO satellites makes the content placement and request assignment problem fundamentally different from prior efforts. We thus build a graph to model the time-varying dynamic satellite-cloud topology.

Vertex set. In terrestrial cloud-CDNs, content providers place the replica of contents on geo-distributed cloud data centers owned by cloud operators. Let Src denote the source/origin server of the content provider. We assume that there are M available cloud data centers offered by the terrestrial cloud operator for caching contents, *i.e.,* $C = \{C_1, C_2, ..., C_M\}$.

In addition, contents can also be cached by LEO satellites, and we assume the constellation consists of N LEO satellites orbiting at the orbital altitude H. The satellite set is denoted as $S = \{S_1, S_2, ..., S_N\}$. Further, to describe the high-dynamicity of LEO satellites, we assume T is the *synodic period* of the given constellation, which refers to the repeated cycles for the satellites as observed from the Earth surface. Assume a period T is split into multiple time slots. We extend the satellite set S to $\mathscr{S} = \{\mathscr{S}^1 \cup \mathscr{S}^2, ..., \cup \mathscr{S}^T\}$, $\mathscr{S}^t = \{S_{1t}, S_{2t}, ..., S_{Nt}\}$, where S_{it} denotes the snapshot of ith satellite in time slot t.

As we have introduced previously, emerging cloud providers may deploy ground stations close to their data centers (*e.g.,* Amazon Ground Station Services [21] and Azure Orbital [4]) to provide on-demand ground communication that can connect cloud data centers to mega-constellations. Therefore, each cloud is also accessible via time-varying space routes constructed by LEO satellites. We thus make a logic snapshot of the cloud data center set C in different slots, and we set $CS^t = \{CS_{1t}, CS_{2t}, ..., CS_{Mt}\}$, $CS = \{CS^1 \cup CS^2, ..., \cup CS^T\}$ as the set of clouds accessible via satellites. CS^t refers to the set of all snapshots of M cloud sites that are accessible via satellite paths in slot t, and more specifically CS_{it} indicates the cloud data center i that is accessible via satellite paths in slot t. Physically, C_i and CS_{it} refer to the same cloud data center, while logically the total cloud set is $\mathscr{C} = \{C \cup CS\}$. We denote the snapshot of clouds in slot t as: $\mathscr{C}^t = \{C \cup CS^t\}$.

CDNs receive and dispatch geographical user requests, which are typically wrapped by HTTP GET/POST requests. In practice, the request dispatch is generally implemented via DNS-based or anycast-based client-server mappings [16], [59]. To model the user distribution, we assume the Earth surface is discretized into non-overlapped grid-like regions, and there are J regions in total, and use $\mathscr{R} = \{R_1, R_2, ..., R_J\}$ to describe the set of all serving regions. User requests are dispatched based on their regions. Summarily, assume Src is the source server, the vertex set of the satellite-cloud topology graph in slot t can be denoted as: $V_t = \{Src \cup \mathscr{S}^t \cup \mathscr{C}^t \cup \mathscr{R}\}$. Further, the vertex set in a period T is accordingly denoted as: $V_T = \{V_1 \cup V_2 \cup ... \cup V_T\}$.

Edge set. For every two different nodes i and j in the vertex set V_T, there might be an undirected edge between them (*i.e.,* (i, j)). We denote the edge set in a period T as E_T. Specifically, an edge connecting two cloud/satellite nodes indicates that there is a data transmission path between them, over either the terrestrial or space network. An edge between a cloud/satellite node and a region indicates that user requests from this region can be assigned to the cloud/satellite node. No edges exist between two different regions as they represent end users. The source server is similar to a cloud node.

Each edge is associated with a $cost_{traffic}(i, j)$ property, which indicates the cost of transferring one data unit between i and j. In addition, if i or j is a region, the edge (i, j) will be associated with a $latency(i, j)$ property, which refers to the access delay (*e.g.,* RTT) between users in this region and the corresponding cache server (*e.g.,* a cloud or a satellite). The concrete value of $cost_{traffic}(i, j)$ and $latency(i, j)$ depends on the related vertex type, described as follows.

- (i) For cloud nodes $i, j \in \mathscr{C}$ in arbitrary timeslot, the data traffic between them is transferred over the terrestrial Internet. We have $cost_{traffic}(i, j) = 0$, if i and j are physically equal, *e.g.,* $i = C_k$ and $j = CS_{kt}$. Otherwise, $cost_{traffic}(i, j) = c$, *i.e.,* transferring one unit data between i and j costs \$$c$.
- (ii) For cloud $i \in \mathscr{C}$ and satellite $j \in \mathscr{S}$ in arbitrary timeslot, the edge connecting them can be established by the shortest-path source-routing approach [36] via a sequence of LEO satellites and a ground station in practice. Consider that data transmission over space path may consume more precious inter-satellite and ground communication resources, we denote $cost_{traffic}(i, j) = \alpha \cdot c$, $i \in \mathscr{C}$ and $j \in \mathscr{S}$, where α is an amplification factor, and $\alpha > 1$.
- (iii) For satellite nodes $i, j \in \mathscr{S}, (i \neq j)$ in arbitrary timeslot, we set $cost_{traffic}(i, j) = \beta \cdot c$. Similar to α, β is an amplification factor as data transmission between satellites might be more costly than that in terrestrial Internet. ($\beta > 1$)
- (iv) For region $i \in \mathscr{R}$ and cloud $j \in \mathscr{C}$, $cost_{traffic}(i, j) = c$, if $j \in C$, and $cost_{traffic}(i, j) = \alpha \cdot c$ if $j \in CS$, as leveraging space routes to provide content access to a cloud site is more costly. In addition, assume users in the same

region perceive similar latencies to the same cache server, and $latency(i,j)$ is the region-to-server access latency between region i and server j. This value indicates how fast the cache server j can response to requests from region r.

- (v) For region $i \in \mathscr{R}$ and satellite $j \in \mathscr{S}$, (i,j) indicates that requests from r can be assigned to the satellite cache server j. We set $cost_{traffic}(i,j) = \alpha \cdot c$. Specifically, if $j = S_{kt}$, $latency(i,j)$ refers to the access latency between region i and satellite k in slot t. Note that as LEO satellites move during the orbit, the latency between a region and a satellite cache may accordingly change over time.

With definitions related to vertexes and edges, we define the connectivity graph in a period T as $G_T =< V_T, E_T >$.

2) Formulating the cost-effective content distribution problem over the satellite-cloud topology: Given the time-varying topology above, next we formulate the cost-effective content distribution problem based on the satellite-cloud model. Formally, the problem can be converted to forming a distribution graph (DG) upon the satellite-cloud topology G_T. Assume $\lambda(i,j)$ is a binary value and $\lambda(i,j) = 1$ indicates that edge (i,j) is selected in DG. Assume $x(i)$ is a sever selection indicator and $x(i) = 1$ if vertex i is selected in DG. Then a DG can be presented as $DG =< V_{DG}, E_{DG} >$, where $x(i) = x(j) = 1, \lambda(i,j) = 1, \forall i,j \in V_{DG}, (i,j) \in E_{DG}$, and $V_{DG} \subset V_T, E_{DG} \subset E_T$.

DG reveals how to push and place contents on a collection of selected cache servers, and how to assign requests from a certain region to a cache server. First, all nodes in V_{DG} are selected as cache servers, and have to store the content replicas. Second, assume r is a region node, then an edge $(r,j) \in E_{DG}, j \in \{C \cup CS^t \cup S^t\}$ connecting a region and a server indicates the time-varying request assignment in the period T. Specifically, let $a(r,t)$ denote the cache server assigned for region r in slot t. $a(r,t)$ is calculated as follows: (i) if in slot t there is an edge in E_{DG} connecting r and a server in CS^t or S^t, then in slot t requests from r will be assigned to the cache server via space routes; (ii) else if in slot t there is only an edge in E_{DG} connecting r to a terrestrial cloud in C, then requests from r will be assigned to the connected cloud via terrestrial routes; (iii) otherwise, no connectable nodes exist for r in slot t, and $a(r,t) = \varnothing$. Therefore, the problem is converted to finding a sub-graph DG from G_T in period T, while satisfying several constraints as described below.

C1: Each region has to be assigned to a cache server in each time slot. The distribution graph DG should guarantee that users in every region are assigned to a cache server during the period T, *i.e.,* $a(r,t) \neq \varnothing, \forall r \in \mathscr{R}, \forall t \in T$.

C2: For each selected cache server, there should be at least one path from the source server to the cache server. Originally, contents are generated on the source server, and have to be pushed to each selected cache server (*i.e.,* where $x(i) = 1$). Therefore, in DG there should be a distribution path from the source server to every cache node.

C3: Request assignments for each region should satisfy a latency requirement specific to various applications. As many applications have a latency requirement to sustain good user experience, in each period T, the average user-perceived latency between each region and its assigned cache server is expected to be lower than a threshold $Thd_{latency}$. Specifically, assume the user-perceived latency for region r in slot t is: $latency(r, a(r,t))$, and thus the latency constraint can be formulated as: $\sum_t \frac{latency(r,a(r,t))}{T} \leq Thd_{latency}, \forall t \in T$.

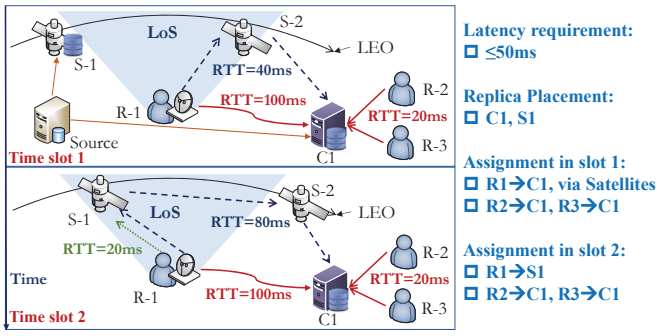

Fig. 4: An example of distributing contents from the source server and corresponding assignment.

Figure 4 plots an example showing a process of content placement and request assignment. Assume there are three regions (R1-3), one cloud site (C1), and two LEO satellite (S1-2), in two time slots. In each time slot, the RTT from R_2 and R_3 to cloud C_1 is about 20ms, which is below the latency requirement 50ms. In slot 1, the RTT between R_1 and C_1 is 100ms via the terrestrial network, and is 40ms via satellites. In slot 2, as S_2 moves out of the transmission range of R_1, the space route from R_1 to C_1 updates, and the RTT between R_1 and C_1 over satellites increases from 40ms to about 80ms. Based on predictable conditions, a viable solution to guarantee the latency requirement is to push and place replica on S_1 and C_1, and: (i) assign requests from R_2 and R_3 to C_1 in each slot; (ii) assign requests from R_1 to C_1 via satellites in slot 1, and (iii) assign requests from R_1 to S_1 in slot 2.

Cost analysis. Note that both the content placement and request assignment operations may involve cost for content providers. In the placement process, distributing contents from the source server to all cache servers consumes bandwidth in terrestrial and space networks, with a cost of $CT_1 = \sum_{i,j} cost_{traffic}(i,j) * \lambda(i,j) * W, \forall i,j \in \{Src \cup \mathscr{C} \cup \mathscr{S}\}$, where W is the content size. Caching contents on clouds or satellites also involves additional storage cost, and we denote $cost_{storage}(i)$ as the cost of storing one data unit on cache server i. We set $cost_{storage}(i) = d$ if i is a cloud server, and $cost_{storage}(i) = \gamma * d$ if i is a satellite, where $\gamma \geq 1$ indicates that on-broad resource is more expensive as compared to terrestrial clouds. Hence the total storage cost is calculated as: $CT_2 = \sum_k cost_{storage}(k) * x(k) * W, k \in \{\mathscr{C} \cup \mathscr{S}\}$.

Let $req(r,t)$ denote the total number of user requests from region r in slot t. Then the cost of the assignment process is involved by consuming bandwidth between users and assigned cache servers. Roughly, the assignment cost can be calculated as: $CT_3 = \sum_{r,t} cost_{traffic}(r, a(r,t)) * req(r,t) * W, \forall r \in \mathscr{R}, t \in T$. Collectively the total cost in a scheduling period T can be formulated as $CT = CT_1 + CT_2 + CT_3$, where CT_1

is the bandwidth cost of distributing source contents to each selected cache server, CT_2 refers to the storage cost of all cache servers, and CT_3 indicates the bandwidth cost of serving user requests at runtime.

Cost-effective content distribution problem. Typically, content providers may have their own budget of exploiting cloud/satellite infrastructures to distribute contents. The cost-effective content distribution problem in our StarFront framework is to find a distribution graph DG from G_T that minimizes the total cost CT, while satisfying all the above constraints $C1, C2$ and $C3$. The cost-effective problem is essentially an instance of the Integer Linear Programming (ILP) problem. We implement the ILP formulation and solve it with Python MIP [9]. The typical processing time of the implementation takes minutes to hours as the number of satellites and user requests increases, which is too long to be practical. This motivates us to design and implement a more efficient and applicable solution.

D. Judicious Replica Placement and Request Assignment

We propose a heuristic to solve the content distribution problem efficiently, with the key idea of judiciously exploring a proper cache server assignment that satisfying the application level latency requirement, while incurring minimal bandwidth and storage cost. In particular, exploiting cache servers to serve end users typically involves three portions of cost. Assume s is the cache server assigned for region r, and recall the cost in each step of a content distribution process. First, the replica has to be pushed from another server that has stored the replica to s. This step consumes inter-cloud/satellite bandwidth cost. Second, provisioning replica on clouds or satellites also incurs storage cost. For each selected cache server, the above two steps only need to be executed once at the beginning of a period T. Finally, during the period T and in every time slot, serving requests from region r also involves bandwidth cost between s and r. Formally, let $Cost_{assign}(r, s)$ denote the cost estimation for assigning requests from r to s.

Algorithm 1 shows the details of StarFront's replica placement and request assignment algorithm. In each period T, StarFront's controller (Figure 3) first generates the connectivity graph G_T based on the cloud distribution and predictable trajectory of satellites during the current period. Then the controller runs the algorithm to decide how to push contents to each selected cache servers and properly assign user requests. For each region r, StarFront first searches the set of all available cloud or satellite servers (*i.e.,* $Candidate$) that can meet the access latency requirement (line 7-12). Among all entries in $Candidate$, StarFront greedily picks the server s involving the minimal cost (line 13-14). During the runtime of the algorithm, if $x(s) = 1$, replicas have already been provisioned on s, and the cost only includes the bandwidth cost incurred by using s to serve user requests from r. Otherwise, if $x(s) = 0$, the operation includes additional cost of pushing and storing replicas on the selected cache server.

Algorithm 1 Judicious Placement and Request Assignment.

1: **Input:** Connectivity graph $G_T =< V_T, E_T >$, Latency requirement Thd.
2: **Output:** Distribution graph DG, represented by $x(k), k \in \{\mathscr{C} \cup \mathscr{S}\}$ and $\lambda(i,j), i,j \in V_T$.
3: /* *Greedily assign requests satisfying Thd while minimizing total cost.* */
4: $x(Src) = 1$ /* Original contents on source server. */
5: **for all** region $r \in \mathscr{R}$ **do**
6: /* Find all available servers satisfying Thd. */
7: $Candidate = \varnothing$
8: **for all** $i \in V_T$ **do**
9: **if** $latency(r,i) \leq Thd$
10: $Candidate.add(i)$
11: **end if**
12: **end for**
13: $Selected \leftarrow argmin_{j \in V_T} Cost_{assign}(r,j)$
14: $x(Selected) = 1, \lambda(r, Selected) = 1$
15: **end for**
16: **return** all $x(k)$ and $\lambda(i,j)$.

V. Performance Evaluation

As most emerging mega-constellations are still in their early stage, it is difficult to conduct experiments on live satellite networks. While there exists many prior efforts for network simulation or emulation, existing works either fail to simulate/emulate the high dynamics of LEO satellite (*e.g.,* NS3, Mininet) or they can not support evaluation of realistic content distribution (*e.g.,* [44], [48]). To address this limitations of previous evaluation methodology, we build a testbed to simulate geo-distributed cloud data centers and constellations based on the public orbital data and implement a prototype of StarFront. We further conduct trace-driven simulations to verify the effectiveness of StarFront on cost-effective latency reduction, and leave the systematical evaluation on system-level effects (*e.g.,* delay, loss, system parameters, *etc.*) upon real deployments as our future work.

A. Simulated Satellite-Cloud Integrated Network

At high-level our testbed incorporates a *topology generator* which loads the information of cloud distributions as well as time-varying satellite trajectories to generate the satellite-cloud network topology. Further, the testbed exploits a number of containers running on top of physical machines to simulate the software behaviors of content distribution (*e.g.,* receiving user requests, querying the cache for required data and sending responses back to users).

Topology generator. The topology generator simulates the satellite-cloud integrated architecture as follows. First, it uses the distribution of Amazon AWS cloud sites as the available cloud data centers [22]. We configure the cloud distribution based on Amazon as it has deployed a large number of world-wide cloud sites and recently is deploying ground station services to interconnect clouds and satellites. Second, the topology generator calculates the time-varying satellite trajectory, which includes the LLA positions (*i.e.,* latitude,

Parameter description	Value
Duration	7 days
# of Total Requests	3.9 million
# of Total Bytes Requested	4137 TB
50th/90th Obj Size	256KB / 1.3 GB

TABLE I: Summary of the CDN trace used in the evaluation.

longitude and altitude) of each satellite in every time slot. Specifically, the trajectory information is calculated by third-party orbit computation tool based on the two-line element (TLE) data generated by [11], and is used to estimate the visibility and distance of each satellite from the view of other nodes (*e.g.,* neighbor satellites, ground stations, or terrestrial users). In our experiment, we evaluate STARFRONT under the first shell of SpaceX's Starlink Phase-I [15] and OneWeb [8] constellations as both of them plan to deploy hundreds or thousands of LEO satellites to provide wide-area coverage and Internet services. As of August 2021, the former constellation consists of 1584 satellites in 72 orbital planes with an altitude of about 550km, while the later one is a planned initial 648-satellite constellation at approximately 1200km altitude. The synodic period of Starlink and OneWeb is configured as 5731s and 6557s respectively, based on their public constellation information. Finally, we set the connectivity of each node in the topology based on their related visibility, *i.e.,* a satellite is connectable for a ground station if the satellite moves into the transmission range.

Clouds and satellites simulation. We use Docker containers [6] running on physical machines to support the simulation of cloud/satellite-based cache servers. Specifically, we run a number of Docker containers on each physical machine, and use each container with network software stack to simulate available cloud/satellite servers. Containers are connected to the physical NIC using macvlan [7], which virtualizes a physical NIC into multiple virtual NICs. We use `tc` to control the time-varying RTT, inter-satellite/satellite-ground connectivity and bandwidth of each link. Inter-cloud network conditions are configured based on the measured values from realistic AWS cloud sites. The connectivity and performance of satellites are configured based on the results characterized by [48].

Dataset and request generator. Our evaluation leverages a real-world CDN trace collected from a commercial cloud CDN operator on February 24, 2015, containing 552 thousands flow records in total. Table I describes the details of the selected trace. Moreover, we write a request generator to simulate user clients. It extracts information from the trace and generates HTTP requests to fetch object data. Each HTTP request issued by end users is processed as follows. First, the user issues a DNS query to the location DNS server. Second, the DNS server returns the IP address of the assigned cache server (one of the Docker container) to the user. Finally the client sends a request to the cache server to fetch the content data.

B. STARFRONT *Prototype*

STARFRONT **controller.** The controller of STARFRONT is implemented in around 1100 lines of Python codes. Periodically, the controller reads the satellite location information and the historical network performance information, and calculates the decisions for content placement and request assignment. Content replicas are then pushed to the cache servers via HTTP connections, following the calculated decision. We follow the pricing policy of existing cloud and satellite operators (*e.g.,* CloudFront [1] and ViaSAT [18]) to estimate the cost function of content delivery via clouds or satellites.

STARFRONT **cache servers.** We have implemented the STARFRONT cache based on Apache Traffic Server (ATS) [3]. ATS is a multi-threaded, event-based, modular, high-performance cache and forward proxy server, written in C++. ATS is distributed as a commercial product and has been used in many production-level systems. We modified ATS to connect to STARFRONT controller and execute the placement decision.

Next, evaluations in this section aim at answering the following two questions: (i) can STARFRONT satisfy various latency requirements of geo-distributed users as compared with other state-of-the-art content distribution approaches under representative CDN traces and constellation patterns? and (ii) what is the corresponding cost of using STARFRONT?

C. *Verifying the Ability of Satisfying Latency Requirements*

We compare the latency reduction on global content distribution of four different strategies: (i) the state-of-the-art low-latency content placement and assignment scheme in existing cloud CDNs *(denoted as Cloud-based SoA)* (*e.g.,* TailCutter [47], GRP [29], CosTLO [58] *etc.*); (ii) STARFRONT, our proposed framework that judiciously exploits cloud and satellite servers to place contents and assign user requests to proper cache servers to satisfy the latency requirements of various applications, while minimizing the operational cost. In addition, to comprehensively understand the incremental effectiveness of integrating LEO satellites to cloud-based CDNs, we evaluate STARFRONT under two specific configurations: (iii) replicas are cached on cloud servers, and users can only access clouds via satellite paths (*i.e.,* $C = \varnothing, S = \varnothing$ but $CS \neq \varnothing$ in Section IV-C); and (iv) replicas are cached on cloud servers, and can be fetched by user through terrestrial or satellite paths (*i.e.,* $S = \varnothing$, but $C \neq \varnothing$ and $CS \neq \varnothing$ in Section IV-C). Strategy (iii) and (iv) refer to the methodology that only exploiting satellite networks to extend the connectivity of terrestrial clouds, without using the storage capability of satellites to cache contents in space. We denote (iii) as cloud cache accessed by satellite paths (CCS) and denote (iv) as cloud cache accessed by terrestrial and satellite paths (CCTS).

Figure 5 plots the CDF of user-perceived RTTs of different content distribution strategies under various latency requirements. The results of STARFRONT are obtained under the configuration of Starlink constellation. Results with OneWeb are similar and omitted due to the page limit. Since STARFRONT integrates clouds and satellites to store and distribute content globally, it outperforms the state-of-the-art cloud-based strategy by 90.51%/66.63%/52.82%/35.62%/15.45% on average, under the RTT requirements 10ms/30ms/50ms/70ms/100ms respectively. More specifically, we make several observations. First, for stringent RTT requirements (*e.g.,* $\leq$10ms), exploiting the satellite network to accelerate cloud access and even directly

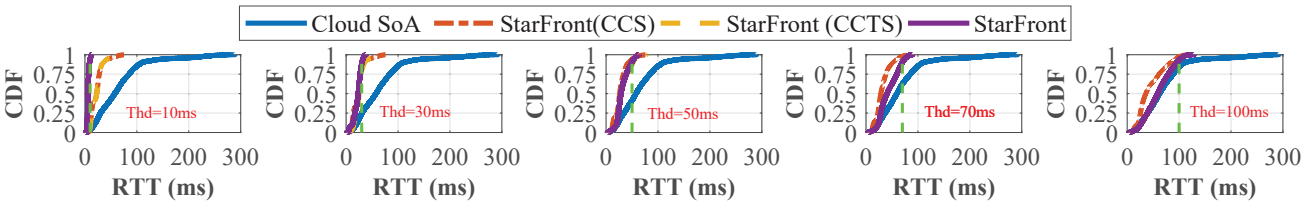

Fig. 5: CDF of RTTs achieved by different strategies under various latency requirements (10/30/50/70/100ms).

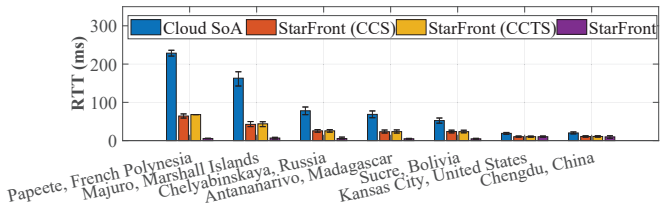

Fig. 6: RTT statistic in seven geo-distributed vantage points.

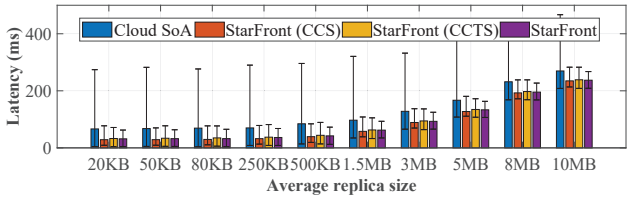

Fig. 7: Latency results under different replica sizes.

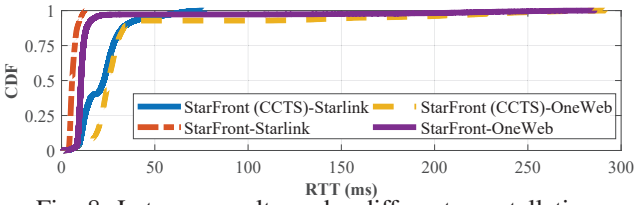

Fig. 8: Latency results under different constellations.

provide cache in space can significantly improve the ability to satisfy the latency requirement for wide-area user requests. This is because incorporating LEO satellites complements terrestrial CDNs and enables low-latency access to cloud and satellite servers from a global perspective. Second, as the required RTT increases (*e.g.,* $10ms \rightarrow 100ms$), the latency performance of StarFront is getting closer to the cloud-based-only approach. This result indicates that under a loose latency constraint StarFront preferably uses more cloud-based resources to save operational costs involved by satellites. Third, caching on satellites can further help reduce the latency, but should inevitably involve much more operational costs. On our further analysis, we find that LEO satellites are more suitable to cache contents that will be requested by international users. This is because LEO satellites inherently have high dynamics, and satellite cache with regional contents may suffer from low cache utilization as it orbits the Earth in high velocity.

Further, we turn our focus on the RTTs on a set of geo-distributed regions. Figure 6 shows the latency comparison under different content distribution strategies for a collection of vantage points around the world. We observe that the latency gain achieved by StarFront differs in different regions. For users in remote or under-developed areas, StarFront can achieve much more latency improvement since the cloud deployment and terrestrial network infrastructure in these regions might be underserved, and LEO satellites extend the availability and performance of terrestrial cloud platforms. Specifically, StarFront reduces more than 90% RTT for users in remote regions such as Papeete and Majuro, as compared to the cloud-only strategy. For users in populated areas like Kansas City and Chengdu, all strategies achieve comparable latency results due to the sufficient deployment of nearby cloud infrastructures.

D. Latency Reduction under Various Replica Sizes

Replicas from the content providers may have different object sizes in practice, according to the concrete application type (*e.g.,* static texts, files or video clips, *etc.*). We assess the content access latency indicating how fast the requested object can be delivered to users under various replica sizes. As shown in Figure 7, we observe that smaller requests with lower object size prone to achieve more latency reduction, as compared to other larger requests. The major benefit of StarFront is to exploit emerging LEO satellites to push contents more close to users and realize lower client-to-content RTT. On the deeper analysis, we find that for small requests the total content access latency is dominated by the RTT, and the content access latency is jointly affected by the achievable throughput and RTT, and small requests are responded faster under low RTT situations.

E. Latency Reduction under Different Constellation Patterns

Different satellite operators have their specific constellation designs, which differ in orbital parameters (*e.g.,* orbital altitude, inclination, *etc.*), and the complete constellation requires a long time to fully deploy. Next we examine the latency reduction under different constellation patterns. Specifically, in our experiment we compare the latency under two state-of-the-art constellations Starlink and OneWeb. As shown in Figure 8, we find that StarFront associated with Starlink can achieve lower latency as compared to OneWeb. The reason is threefold. First, OneWeb satellites are working on a higher altitude as compared to Starlink, and thus it suffers higher propagation delay when working as a cache server or providing network connectivity to a terrestrial cloud. Second, OneWeb satellites do not have inter-satellite data links, hence it limits the latency improvement when using satellite to construct space routes to extend the accessibility of cloud servers. Third, as the Starlink constellation consists of more LEO satellites than OneWeb, and if those satellites have adequate storage capability to cache contents, StarFront associated with Starlink can obtain higher latency gain since a denser constellation enables more cache servers and diverse low-latency space routes for fetching contents from a cloud.

Cost($/GB) Method / (α, β, Thd)	Cloud SoA	SF (CCS)	SF (CCTS)	SF
$< 2, 2, 10ms >$	0.108	0.208	0.203	0.250
$< 2, 2, 50ms >$	0.105	0.202	0.162	0.163
$< 2, 5, 10ms >$	0.107	0.211	0.183	0.240
$< 5, 5, 10ms >$	0.108	0.507	0.485	0.611
$< 5, 5, 50ms >$	0.104	0.503	0.299	0.398

TABLE II: Content distribution cost under different pricing policies and latency requirements.

F. Operational Costs

Finally, we evaluate the operational cost, including both the storage and bandwidth involved by the content distribution process. Table II summarizes the cost (sum of storage and bandwidth) of content distribution under different strategies (*i.e.,* the average cost of distributing one GB data to users), with various configurations of the coefficient α, β and Thd. As shown in Table II, the cost increases as we enlarge the cost coefficient α and β. As compared to cloud-only strategies, StarFront achieves lower latency at the cost of more content distribution fees. While more expensive, the cost might not be unacceptable, and should be worth for high-priority regions or tasks with very stringent latency requirements.

VI. Related Work

We briefly discuss the related works in this section.

Replica placement and request assignment in CDNs. A considerable number of previous research has studied the replica placement or request assignment problem in cloud-based CDNs [20], [29], [41], [43], [47], [49], [53], [58], [59]. CostTLO [58] combines the use of issuing redundant requests to fetch contents from different cloud storage nodes. NetSession [61] is a peer-assisted CDN system that leverages both dedicated, centrally managed infrastructures and clients to distribute contents. NetSession can deliver several of the key benefits of both infrastructure-based and peer-to-peer CDNs. The fundamental difference between StarFront and existing cloud-based works is that: StarFront is a framework exploiting network and storage resources on clouds *(static)* and LEO satellites *(dynamic)* cooperatively to optimize the global content delivery efficiency. Different from prior efforts (*e.g.,* NetSession) StarFront characterizes the high mobility of LEO satellites, and leverages their predicted trajectory to assist the calculation of content placement and request assignment decisions. As shown in the evaluation, through exploiting LEO satellites equipped with storage and high-datarate ISLs, StarFront can improve the delivery efficiency of CDNs in a cost-effective manner, especially for the large population in remote areas.

Exploring networks constructed upon "NewSpace" constellations. Since the design of StarFront is partially based on emerging LEO constellations, our work is inspired by a number of recent studies on characterizing and understanding the network performance of "NewSpace" constellations [24], [25], [30], [31], [35]–[38], [44], [45], [48], [55]. Several works have studied the routing algorithm in satellite networks. For example, to understand the latency properties of LEO satellite networks, the author in [36] builds a simulator to evaluate how to use the laser links to enable low-latency communication over satellites. Similarly, a ground-relay based routing strategy is proposed in [37]. Authors in [26] propose a path-aware networking architecture to optimize Internet routing over an integrated satellite and terrestrial network. Moreover, Motifs [26] is proposed to dynamically build network connectivities in space, tackling the high dynamism in LEO satellites. All these works complement our work.

In-Orbit computing. Recent works also explore the feasibility of leveraging the improved computation capability in emerging nanosatellite constellations to build a new class of computer system [25], [32], [46]. Authors in [25] qualitatively examine the opportunities and challenges of in-orbit computing. An orbital edge computing (OEC) [32] architecture is proposed to address the limitation of existing "bent-pipe" architecture and optimize the edge processing latency. Instead of focusing on nanosatellites only, our work proposes a cooperatively architecture that jointly exploits the resource on both terrestrial cloud data centers and LEO satellite to judiciously improve the performance of content distribution on a global scale.

Caching algorithms. The algorithm that determines how to update or replace objects cached in each edge node is also an important issue for CDNs. The state-of-the-art algorithms are typically heuristic-based (*e.g.,* Least-Recently-Used and its variants [23], [33], [39], [51]) or learning-based [52], [57]. Learning Relaxed Belady (LRB) is a recently proposed CDN cache design which leverages a novel metric *good decision ratio* to optimize the *byte miss ratios* in cache-like systems. These studies focusing on the caching strategy in dedicated edge nodes complement our work.

VII. Conclusion

This paper investigates the problem of constructing low-latency, cost-effective CDNs upon emerging LEO satellites and clouds. We analyze and quantify the potential benefits, feasibility and challenges of a satellite-cloud integrated CDN for improving the content access latency. We present StarFront, a cost-effective framework that takes dynamic network topology, workload distribution, and pricing policy from satellite/cloud operators as the input, and optimizes the access latency of content distribution. Trace-driven evaluations show that by cooperatively and judiciously placing replicas on satellites and clouds, StarFront can satisfy various latency requirements from applications with acceptable cost, as compared to existing cloud-based approaches.

VIII. Acknowledgements

We thank our shepherd Ignacio Castro and the anonymous ICNP reviewers for their comments and suggestions. This work was supported in part by National Key Research and Development Program of China (No. 2020YFB1806001), NSFC (No. 61832013), China Postdoctoral Science Foundation (No. 2021M691786) and Tsinghua University-China Mobile Communications Group Co., Ltd. Joint Institute.

References

[1] Amazon cloudfront pricing. https://aws.amazon.com/cloudfront/pricing/.

[2] Amazon s3 pricing. https://aws.amazon.com/s3/pricing/.

[3] Apache traffic server. https://trafficserver.apache.org/.

[4] Azure orbital: Satellite ground station and scheduling service connected to azure for fast downlinking of data. https://azure.microsoft.com/en-us/services/orbital/.

[5] Cisco visual networking index (vni), complete forecast update (2017-2022). https://www.cisco.com/c/dam/m/en_us/network-intelligence/service-provider/digital-transformation/knowledge-network-webinars/pdfs/1213-business-services-ckn.pdf.

[6] Docker container. https://www.docker.com/.

[7] Get started with macvlan network driver. https://docs.docker.com/network/macvlan/.

[8] Oneweb. https://www.oneweb.world/.

[9] Python mip (mixed-integer linear programming) tools. https://pypi.org/project/mip/.

[10] Ripe atlas measurement platform. https://atlas.ripe.net/.

[11] Space-track.org. https://www.space-track.org/.

[12] Spacebelt aims to store data in satellites. https://blocksandfiles.com/2020/04/21/spacebelt-store-data-in-satellites-analysis/.

[13] Spacebelt cloud constellation corporation. http://spacebelt.com/.

[14] Spacex's starlink satellite internet service is priced at 99 dollars per month. https://www.cnbc.com/2020/10/27/spacex-starlink-service-priced-at-99-a-month-public-beta-test-begins.html.

[15] Starlink. https://www.starlink.com/.

[16] Use dns policy for geo-location based traffic management with primary servers. https://docs.microsoft.com/en-us/windows-server/networking/dns/deploy/primary-geo-location.

[17] Viasat: high-speed internet for your home or business. https://www.viasat.com/internet.

[18] Viasat internet plans and pricing. https://www.satelliteinternet.com/providers/viasat/.

[19] World internet usage and population statistics 2020. https://www.internetworldstats.com/stats.html.

[20] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: Automated data placement for geo-distributed cloud services. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, page 2, USA, 2010. USENIX Association.

[21] Amazon. AWS Ground Station. https://aws.amazon.com/ground-station/, 2021. [Online; accessed 30-August-2021].

[22] Amazon. Global Infrastructure. https://aws.amazon.com/about-aws/global-infrastructure/, 2021. [Online; accessed 30-August-2021].

[23] H. Bahn, K. Koh, S. H. Noh, and S. Lyul. Efficient replacement of nonuniform objects in web caches. *Computer*, 35(6):65–73, 2002.

[24] D. Bhattacherjee, W. Aqeel, I. N. Bozkurt, A. Aguirre, B. Chandrasekaran, P. B. Godfrey, G. Laughlin, B. Maggs, and A. Singla. Gearing up for the 21st century space race. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, HotNets'18, pages 113–119, New York, NY, USA, 2018. Association for Computing Machinery.

[25] D. Bhattacherjee, S. Kassing, M. Licciardello, and A. Singla. In-orbit computing: An outlandish thought experiment? In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, HotNets '20, page 197–204, 2020.

[26] D. Bhattacherjee and A. Singla. Network topology design at 27,000 km/hour. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 341–354, 2019.

[27] I. N. Bozkurt, A. Aguirre, B. Chandrasekaran, P. B. Godfrey, G. Laughlin, B. Maggs, and A. Singla. Why is the internet so slow?! In *International Conference on Passive and Active Network Measurement*, pages 173–187. Springer, 2017.

[28] I. Castro, J. C. Cardona, S. Gorinsky, and P. Francois. Remote peering: More peering without internet flattening. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 185–198, 2014.

[29] F. Chen, K. Guo, J. Lin, and T. La Porta. Intra-cloud lightning: Building cdns in the cloud. In *INFOCOM*. IEEE, 2012.

[30] I. del Portillo, B. Cameron, and E. Crawley. Ground segment architectures for large leo constellations with feeder links in ehf-bands. In *2018 IEEE Aerospace Conference*, pages 1–14, 2018.

[31] I. del Portillo, B. G. Cameron, and E. F. Crawley. A technical comparison of three low earth orbit satellite constellation systems to provide global broadband. *Acta Astronautica*, 159:123–135, 2019.

[32] B. Denby and B. Lucia. Orbital edge computing: Nanosatellite constellations as a new class of computer system. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 939–954, 2020.

[33] B. Fan, D. G. Andersen, and M. Kaminsky. Memc3: Compact and concurrent memcache with dumber caching and smarter hashing. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 371–384, 2013.

[34] H. Fenech, S. Amos, A. Tomatis, and V. Soumpholphakdy. High throughput satellite systems: An analytical approach. *IEEE Transactions on Aerospace and Electronic Systems*, 51(1):192–202, 2015.

[35] G. Giuliari, T. Klenze, M. Legner, D. Basin, A. Perrig, and A. Singla. Internet backbones in space. *SIGCOMM Comput. Commun. Rev.*, 50(1):25–37, Mar. 2020.

[36] M. Handley. Delay is not an option: Low latency routing in space. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, HotNets '18, page 85–91, New York, NY, USA, 2018. Association for Computing Machinery.

[37] M. Handley. Using ground relays for low-latency wide-area routing in megaconstellations. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, HotNets '19, page 125–132, New York, NY, USA, 2019. Association for Computing Machinery.

[38] Y. Hauri, D. Bhattacherjee, M. Grossmann, and A. Singla. "internet from space" without inter-satellite links. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, HotNets '20, page 205–211, 2020.

[39] X. Hu, X. Wang, Y. Li, L. Zhou, Y. Luo, C. Ding, S. Jiang, and Z. Wang. Lama: Optimized locality-aware memory allocation for key-value cache. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 57–69, 2015.

[40] H. Huang, S. Guo, and K. Wang. Envisioned wireless big data storage for low-earth-orbit satellite-based cloud. *IEEE Wireless Communications*, 25(1):26–31, 2018.

[41] Y. Huang, Z. Li, G. Liu, and Y. Dai. Cloud download: using cloud utilities to achieve high-quality content distribution for unpopular videos. In *MM*. ACM, 2011.

[42] X. Jia, T. Lv, F. He, and H. Huang. Collaborative data downloading by using inter-satellite links in leo satellite networks. *IEEE Transactions on Wireless Communications*, 16(3):1523–1532, 2017.

[43] G. Joshi, Y. Liu, and E. Soljanin. On the delay-storage trade-off in content download from coded distributed storage systems. *JSAC*, 2014.

[44] S. Kassing, D. Bhattacherjee, A. B. Águas, J. E. Saethre, and A. Singla. Exploring the "internet from space" with hypatia. In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, page 214–229, 2020.

[45] T. Klenze, G. Giuliari, C. Pappas, A. Perrig, and D. A. Basin. Networking in heaven as on earth. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks, HotNets 2018, Redmond, WA, USA, November 15-16, 2018*, pages 22–28. ACM, 2018.

[46] V. Kothari, E. Liberis, and N. D. Lane. The final frontier: Deep learning in space. In *HotMobile '20: The 21st International Workshop on Mobile Computing Systems and Applications, Austin, TX, USA, March 3-4, 2020*, pages 45–49. ACM, 2020.

[47] Z. Lai, Y. Cui, M. Li, Z. Li, N. Dai, and Y. Chen. Tailcutter: Wisely cutting tail latency in cloud cdn under cost constraints. In *INFOCOM*. IEEE, 2016.

[48] Z. Lai, H. Li, and J. Li. Starperf: Characterizing network performance for emerging mega-constellations. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, 2020.

[49] Z. Li, Y. Huang, G. Liu, F. Wang, Z.-L. Zhang, and Y. Dai. Cloud transcoder: Bridging the format and resolution gap between internet videos and mobile devices. In *NOSSDAV*. ACM, 2012.

[50] N. Pachler, J. J. G. Luis, M. Guerster, E. Crawley, and B. Cameron. Allocating power and bandwidth in multibeam satellite systems using particle swarm optimization.

[51] S.-y. Park, D. Jung, J.-u. Kang, J.-s. Kim, and J. Lee. Cflru: a replacement algorithm for flash memory. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, pages 234–241, 2006.

[52] Z. Song, D. S. Berger, K. Li, and W. Lloyd. Learning relaxed belady for content distribution network caching. In *17th USENIX Symposium*

on Networked Systems Design and Implementation (NSDI 20), pages 529–544, 2020.

[53] A. Sundarrajan, M. Kasbekar, R. K. Sitaraman, and S. Shukla. Midgress-aware traffic provisioning for content delivery. In *2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020*, pages 543–557. USENIX Association, 2020.

[54] D. Vasisht and R. Chandra. A distributed and hybrid ground station network for low earth orbit satellites. HotNets '20, page 190–196, 2020.

[55] F. Vidal, H. Legay, G. Goussetis, M. Garcia Vigueras, S. Tubau, and J.-D. Gayrard. A methodology to benchmark flexible payload architectures in a megaconstellation use case. *International Journal of Satellite Communications and Networking*, 2020.

[56] O. Vidal, G. Verelst, J. Lacan, E. Alberty, J. Radzik, and M. Bousquet. Next generation high throughput satellite system. In *2012 IEEE First AESS European Conference on Satellite Telecommunications (ESTEL)*, pages 1–7, 2012.

[57] M. V. Wilkes. Slave memories and dynamic storage allocation. *IEEE Transactions on Electronic Computers*, (2):270–271, 1965.

[58] Z. Wu, C. Yu, H. V. Madhyastha, and U. Riverside. Costlo: Cost-effective redundancy for lower latency variance on cloud storage services. In *NSDI*. USENIX, 2015.

[59] J. Xue, D. Choffnes, and J. Wang. Cdns meet cn an empirical study of cdn deployments in china. *IEEE Access*, 5:5292–5305, 2017.

[60] H. Zech, F. Heine, D. Tröndle, S. Seel, M. Motzigemba, R. Meyer, and S. Philipp-May. LCT for EDRS: LEO to GEO optical communications at 1,8 Gbps between Alphasat and Sentinel 1a. In E. M. Carapezza, P. G. Datskos, C. Tsamis, L. Laycock, and H. J. White, editors, *Unmanned/Unattended Sensors and Sensor Networks XI; and Advanced Free-Space Optical Communication Techniques and Applications*, volume 9647, pages 85 – 92. International Society for Optics and Photonics, SPIE, 2015.

[61] M. Zhao, P. Aditya, A. Chen, Y. Lin, A. Haeberlen, P. Druschel, B. Maggs, B. Wishon, and M. Ponec. Peer-assisted content distribution in akamai netsession. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 31–42, 2013.