Krios: Scheduling Abstractions and Mechanisms for Enabling a LEO Compute Cloud

Vaibhav Bhosale
Georgia Institute of Technology
Atlanta, Georgia, USA

Ada Gavrilovska
Georgia Institute of Technology
Atlanta, Georgia, USA

Ketan Bhardwaj
Georgia Institute of Technology
Atlanta, Georgia, USA

ABSTRACT

Low Earth Orbit (LEO) satellites are an important facet of global connectivity providing high speed Internet, cellular, IoT connectivity and so on. Combined with the rich resource availability on each satellite, LEO satellites represent a new, emerging cloud frontier – the LEO Compute Cloud. However, satellite mobility introduces non-trivial challenges when orchestrating applications for a LEO compute cloud, making it harder to deploy applications without increasing the latency and bandwidth costs. In this paper, we identify the concrete challenges in using state-of-the-art terrestrial orchestrators for a LEO compute cloud. We present Krios – a LEO compute cloud orchestration system that hides the complexities introduced by satellite mobility and enables a practical LEO compute cloud. The design of Krios is centered around a novel *LEO zones* abstraction that allows application providers to specify where their applications should be available. Krios provides crucial system support to enable the LEO zones abstraction, ensuring uninterrupted availability of applications in LEO zones via *proactive and predictive application handovers*. Our experimental evaluation of Krios with representative applications demonstrates a practical and efficient LEO compute cloud, without requiring any disruptive changes in applications and with modest system overheads. With Krios, LEO orchestration requires just ~1 application instance at a time to maintain the same availability as what prior work achieves by deploying application instances on all satellites or by performing 6-10 times more frequent expensive handovers.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**.

SoCC '24, November 20–22, 2024, Redmond, WA, USA

ACM ISBN 979-8-4007-1286-9/24/11.
https://doi.org/10.1145/3698038.3698566

KEYWORDS

Low Earth Orbit satellite, LEO Compute Cloud, satellite edge computing, in-orbit computing

ACM Reference Format:
Vaibhav Bhosale, Ada Gavrilovska, and Ketan Bhardwaj. 2024. Krios: Scheduling Abstractions and Mechanisms for Enabling a LEO Compute Cloud. In *ACM Symposium on Cloud Computing (SoCC '24), November 20–22, 2024, Redmond, WA, USA.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3698038.3698566

1 INTRODUCTION

Low Earth Orbit (LEO) satellite networks have transformed global connectivity offering high-speed broadband [13–16, 66], cellular services [49, 70, 77, 79, 101, 103], IoT connectivity [11], as well as augmenting connectivity in a variety of disaster-affected and remote settings [20, 22, 45, 49, 67, 73, 79, 90, 101, 103]. In addition to the large megaconstellations, there are proposals for space datacenter from academia [31], government [80], as well as industry [12]. This has led to LEO satellites, each comprising up to 250 computer nodes [24], to be considered a new, emerging cloud frontier for deploying services such as network functions [59] as well as edge functions similar to terrestrial ISPs [44] and mobile networks [1]. Prior work has studied a diverse set of LEO compute applications including cellular network functions [70, 71], content delivery networks (CDNs) [28, 84] and IoT aggregation [85]. Concretely, we look at two applications in this paper – serving trails at a national park (web serving) and aggregating sensor information from oil rigs (IoT aggregation).

A prerequisite for achieving the desired outcomes when deploying these applications is ensuring they are deployed on satellites that can serve the specific geographic areas they are intended for. This is a difficult task as LEO satellites move at high speeds of over 27,000 km/hr and hence are directly accessible for a maximum of just 4.5 minutes (median 3.5 minutes [30]). However, it is important for applications to be deployed on satellites directly accessible from the users, as the latency and bandwidth cost otherwise increases dramatically (Figure 1). A trivial approach, taken by most of the prior work [40, 41, 84, 85, 87], is to deploy the same application on all satellites such that it is always directly accessible. Another approach taken by prior work is to deploy stateless cellular network functions [70, 71] on all satellites

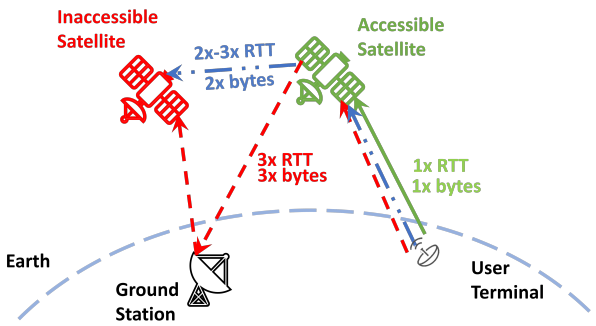


Figure 1: Comparing the bytes consumed and latency incurred for an application deployed on both an accessible (green) and an inaccessible (red) satellite.

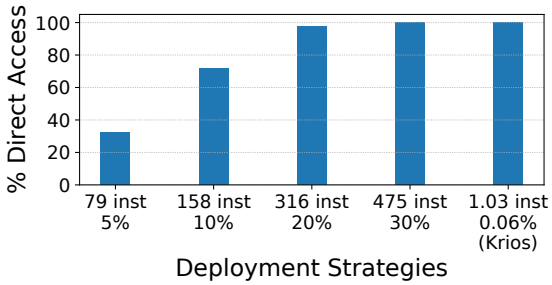


Figure 2: Availability (defined as direct access) of a stateless IoT aggregation application deployed on 5%, 10%, 20%, and 30% satellites in the first Starlink shell during one orbital period (100 mins) compared to just 1.03 instances with Krios.

with the state residing on the user terminals. Our analysis in Figure 2 for a stateless function shows that an application instance needs to run on at least 475 satellites (as opposed to all satellites proposed in prior work), provided they are deployed uniformly (otherwise you need more instances), to ensure continuous availability[1] to a single stationary end user for a 100 minute interval. This gets exacerbated if the application requires state for every instance. Regardless, maintaining at least 475 replicas (and state if applicable) per application is not only a major orchestration challenge but also an inefficient use of fixed satellite resources, severely limiting their utility by limiting the number of applications that can be supported.

Alternatively, prior work [27], including our preliminary findings [28], demonstrated that continuous application availability can be achieved through application handovers, akin to handovers in cellular networks. Handovers in LEO compute are significantly different than cellular handovers, which primarily involve handing over a network connection taking about 30-60 ms [3, 82] and are triggered reactively when a user moves out of range of a (stationary) base station. On the other hand, as the infrastructure is itself moving for LEO compute cloud, in addition to migrating the network connections, handover entails running a new application instance (along with its state) on the new satellite. Depending on the specific scenario, the application (image and the state) may need to be rebuilt or migrated from another satellite. Further, even if the application image and state is available on the new satellite, initializing it takes non-negligible amount of time which needs to be hidden while performing the handover. Empirical network measurements on Starlink suggest handovers moving from reactive [64] (handovers triggered by the satellite moving out of sight) to periodic [57, 78, 95] (handovers occurring periodically every 15 seconds). A plausible explanation for this switch is better resource utilization for satellites (increased elevation angles reduce energy utilization by reducing distance between satellites and users [95]) and performance (packet losses during handovers [64]). We argue that this change only addressed the symptom – reactive handovers lead to packet losses which the frequent 15-second handovers mask, however, they raise other issues like additional bytes on the network due to many (unnecessary) handovers or loss of performance by rebuilding the application (and the state [28]) due to eager handovers, even when the satellite could serve that region for longer [95].

We posit that a key reason why a lot of prior work chooses these periodic or reactive approaches is a result of *missing necessary abstractions* needed to specify where applications need to be available (like terrestrial zones). The continuous motion of the satellites makes it harder for terrestrial orchestrators to identify satellites for deployment, something the zone abstraction helps with terrestrially. Further, even if the orchestrator can determine the correct satellite initially, the orchestrator has *missing systems support* to deal with mobility and ensure availability by accurately identifying deployment targets and triggering handovers in a timely manner. The availability of satellite path models [53] offers a new opportunity to accurately predict the timing and destination of the handovers, instead of relying on signal strength measurements as done in cellular networks. Current orchestration stacks, tailored for a fixed, terrestrial compute infrastructure, lack these elements because they are assume an inherent tight coupling between the orchestrators and the underlying hardware.

In response, we present **Krios** – a software orchestration system that enables application providers to deploy their applications on LEO satellites without having to worry about the mobility of the infrastructure, i.e., the satellite nodes. The key design elements of Krios are:

- A new *software-defined LEO zones* abstraction that allows application providers to specify the geographical region(s) where their application should be available. This abstraction is essential for controlling and managing application placement within the dynamic LEO satellite infrastructure.

[1] In this paper, we define availability by focusing exclusively on application downtime caused by satellite movement to specifically study the impact of satellite motion and observe the gains made by Krios.

LEO zones enable application providers, for the first time to the best of our knowledge, to specify with fine granularity the area of interest they want their application to serve, without requiring any changes in the application deployment workflows.
- Krios introduces the use of satellite path prediction models as a fundamental building block for *proactive and predictive application handovers* onto the physically accessible infrastructure, thus maintaining the geo-stationarity of applications to ensure continuous availability. We design this to not only provide mechanisms to perform handovers, but also to minimize the number of handovers, thus minimizing the bandwidth consumption, latency impact and any downtime due to satellite motion.

The outcome is a new orchestration system for the LEO compute cloud, which *loosens the coupling* between the orchestration stack and the underlying physical infrastructure. Krios can be seamlessly integrated with terrestrial orchestrators, extending their use in space. In summary, this papers makes the following contributions:
- Articulate the non-trivial challenges in scheduling LEO compute functions (§3) to serve specific regions;
- Make the case for a new LEO zones abstraction (§4) and present the design of a new LEO compute orchestration system enabled by it, which offers a practical and effective LEO compute cloud via proactive handovers (§5);
- Experimental evaluation with several LEO compute applications justifies the design choices in Krios and demonstrates its effectiveness to achieve end-to-end benefits – providing similar latency and bandwidth savings compared to prior work while using 400x fewer instances, reducing the frequency of handovers by 6x compared to current Starlink policies, and increasing availability from 5% to 100% compared to current state of the art orchestrators with minimal overheads (§7).

2 BACKGROUND

Lower satellite manufacturing and launch costs [2, 33, 46, 74] have resulted in accelerated deployment of LEO satellites [55]. This has driven the usage of these satellites for a number of use cases such as broadband Internet [13–16, 19, 20, 23, 66, 73, 75, 77], serving disaster affected areas [22, 45, 67, 90], IoT connectivity [11] via communication satellites, imagery via observation satellites [25, 35], climate monitoring [7, 102], and many more. These satellites can vary in size – from smaller 3U (10x10x30 cm) Planet Labs dove satellites to larger 7x3.5 m Starlink v2 satellites. For most of our discussions in this paper, we use the Starlink constellation, which currently operates over 6400 LEO satellites [55]. Starlink currently operates over 170 ground stations [4, 58] and plans to use Google and Microsoft data centers [6, 38] as additional ground stations.

Operating Model. This paper assumes the following model of the LEO ecosystem. The physical infrastructure of LEO constellations comprises of satellites in space and ground stations on Earth. The satellites communicate with the ground stations using wireless radio links using ground satellite links (GSLs). These links generally have Gbps capacities – Starlink GSLs have 18-23 Gbps [94] while Planet Labs GSLs have up to 1.6 Gbps [42]. The satellites communicate with each other using inter-satellite links (ISLs), which could be radio or laser based. Starlink satellites have up to 250 computer nodes onboard every satellites [24]. Similarly, Planet Labs satellites are equipped with GPUs that aid with machine learning computations. Further, observational satellites have GBs/TBs storage to capture the TBs of data they generate every day [9, 43, 88].

A key assumption we make in this paper is that the satellites (when actively serving) are always connected to the ground stations. For the communication satellites, this is true today as all the internet traffic served through these satellites needs to be routed to terrestrial datacenters or points of presence (PoPs) through ground stations using a combination of ISLs and GSLs (the bent-pipe model [69]). Even observational satellites are expected to be equipped with ISLs, ensuring continuous connectivity in the next few years, with successful tests announced by Planet Labs recently [37]. We assume the control plane of the LEO compute orchestration stack is overlaid through the GSLs and ISLs which help a terrestrial orchestrator (e.g., Kubernetes) reach all the satellites.

Target Applications. A lot of prior work has shown LEO compute applications helping to reduce the bandwidth bottlenecks along with reducing latency.

The increased usage of communication satellites [23, 75] has led to bandwidth congestion [17, 36, 60, 62, 76] motivating the need for an edge similar to the use of edge computing terrestrially [18, 26, 54, 92]. Prior work has demonstrated the feasibility of deploying applications on a 'LEO edge', including web servers [28], CDNs [84], IoT aggregation, video conferencing servers [85], etc. While deploying applications on the ground stations does constitute of an edge, it does not lead to reduced latency or bandwidth consumption. Concretely, user terminal to ground station latency can be as high as 150 ms [57] rendering its benefits as an edge ineffective.

On the other hand, the rapid increase in the number of observation satellites and the ensuing growth of collected data has bottlenecked the GSL links, resulting in satellites being unable to transmit all of the generated data [39–41, 96], as done in the bent-pipe mode. Further, sending such large amounts of data results in significant delays of up to a few hours (or even days) in getting insights from this data. To address these challenges, prior work [39–41, 96] has proposed

deploying ML inference on satellites to gain faster insights by reducing of data that needs to be transmitted back.

Regardless of the specific application, it needs to be orchestrated to ensure continuous availability in the desired geographical region. In this paper, our goal is to streamline the application deployment process similar to terrestrial container orchestration systems, including the managed Kubernetes services such as AKS, Amazon EKS, GKE, etc. We next present the specific challenges we have identified that motivated us to design a new system – Krios.

3 TECHNICAL CHALLENGES

Missing Abstractions for Targeted Deployments. A lot of terrestrial applications are designed to benefit from localization, i.e., to exploit the spatial locality of their users. In a terrestrial edge/cloud offering, providers rely on the abstraction of availability zones (or regions) to select one or more zones close to their users where their applications are deployed and are available. However, this abstraction only holds when the underlying physical infrastructure is stationary. In other words, infrastructure is tightly coupled to the zone abstraction. In contrast, in the case of LEO compute, the infrastructure is constantly moving. Therefore, similar tight coupling is not feasible as the satellite nodes move away in a few minutes, rendering the 'infrastructure defined zone' paradigm unusable for LEO compute. Therefore, it is imperative to rethink the abstraction of zones for LEO compute that accounts for this rapid motion of the infrastructure.

Supporting any such notion of zones for application deployment requires a system and policies that can hide the dynamism of the infrastructure. Addressing these challenges forms the next set of technical challenges faced in building the system presented in this paper.

Missing LEO Compute Orchestration System. The main difference between a terrestrial compute cloud and a LEO compute cloud is the intermittent visibility of the LEO satellites. Every satellite is accessible for a maximum of 4.5 minutes [30]. However, the control plane of the current off-the-shelf orchestration stacks is not equipped with the mechanisms necessary to handle this level of dynamism. They are designed to be reactive to changes and rely on monitoring the state of applications (liveness) or infrastructure (failure) to detect when to take corrective actions. Barring failures or capacity issues, an application in a running state is usually not moved to other nodes (even in the cases when applications are migrated due to user mobility, the migration is still reactive to user motion). If a LEO compute cloud uses such reactive mechanisms, it will lead to frequent application downtime as the handover process requires a few minutes to detect and perform any corrective actions [28]. Addressing this requires the orchestration stacks to identify proactive triggers with respect to the dynamism in the infrastructure.

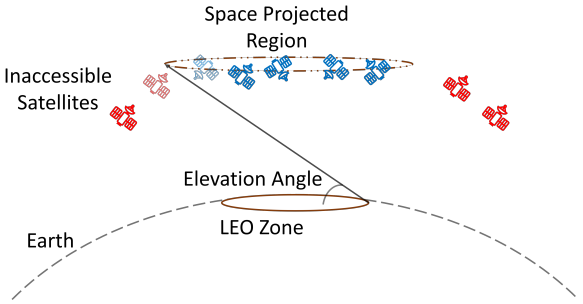


Figure 3: Defining the space projected region for a LEO zones based on elevation angle constraints.

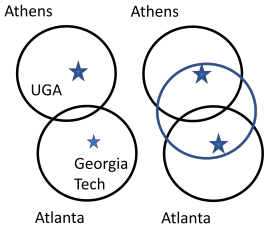


Figure 4: Statically defined vs. LEO zones.

```
metadata:
 name: nginx-pod
 labels:
  app: nginx
 leozone: '37_-122'
 radius: '100'
```

Figure 5: Deployment changes to use Krios.

4 LEO ZONES – THE MISSING ABSTRACTION

We introduce a new *software-defined LEO zones* abstraction that enables defining availability zones based on the location of the end users and not the physical infrastructure. This abstraction of LEO zones is necessary as it allows for deploying applications only on satellite(s) that can actually serve the end users instead of deploying them on all the satellites to contend with the motion of the satellites. At a high level, this serves the same utility as zones in terrestrial edge/clouds and provides an illusion of a geo-stationary infrastructure over moving LEO satellites. This enables leveraging the geographically distributed continuum of compute resources in the form of LEO satellites compared to the sparsely located terrestrial compute infrastructure. Additionally, our definition of LEO zones grants application providers the added flexibility to define their own zones based on their needs, in software.

LEO zones, similarly to the terrestrial edge/cloud zones, are incorporated as part of the application deployment configuration to specify the geographical region of availability. Just like how a terrestrial zone guides the orchestrator to use nodes from a designated cluster, the definition of LEO zones guides the LEO compute orchestrator about which satellites can serve a specific LEO zone and to select the satellite for deploying an application. However, unlike in terrestrial orchestrators, the choice is made not among the nodes known

to a zone *a priori*, but rather from a dynamically determined set of satellite nodes based on their present (and anticipated) geopositioning. The selection is based on the LEO zone's *space projected region*, which is a cone that is calculated based on the allowed elevation angle. The elevation angle determines a LEO satellite's instantaneous accessibility from a given point on Earth and also the duration for which a particular satellite remains accessible. For instance, in Figure 3, the LEO zone definition would restrict the orchestrator to only deploy applications on the blue-colored satellites. We show how the space projected region is calculated from the given LEO zone in Appendix A. While we stick with circular regions for simplicity in this paper, similar calculations can be done for non-trivially shaped regions (Appendix B). Essentially, the space projected region of a LEO zone is the intersection of the space projected regions (i.e., a cone) of all points in the LEO zone.

Although intuitively straight-forward, the LEO zones abstraction is novel and, to the best of our knowledge, the first of its kind. It is different from the way zones have been defined conventionally for terrestrial infrastructure: i.e., instead of the infrastructure defining the availability zones (and the ensuing tight coupling), LEO zones enable application providers to define their own availability zones based on their end users. While one could choose to provide a fixed set of LEO zones when the applications and their users are well understood, it doesn't generalize to the cases when the workloads aren't known apriori. This isn't the case terrestrially where the infrastructure is sparse and hence the fixed set of terrestrial zones is a necessity. Nevertheless, even a fixed set of LEO zones is a new abstraction as they are very different in terms of what they provide and how they are used compared to the conventional zones in a terrestrial cloud/edge.

We show the benefits of such custom-defined LEO zones using an illustrative example. Researchers from two universities, Georgia Tech and University of Georgia (UGA), want to deploy a LEO compute application catering to both campuses. If zones are defined as per terrestrial zones, we can assume that Atlanta and Athens will have their own zones. However, neither of them alone will suffice since they cannot provide continuous coverage to all the intended end users. If the application is deployed in only one zone, users in the other zone will intermittently require multiple hops to reach the satellite, thus leading to increased bandwidth and latency. To mitigate this problem, the same application would need to be deployed in both zones. However, LEO zones would enable the application provider to define a zone that can cover both campuses so that, instead of two instances, a single application instance would suffice (Figure 4).

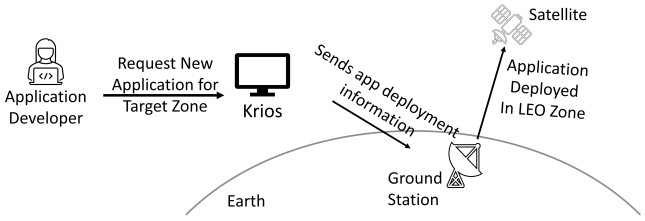


Figure 6: Workflow of application deployment in Krios.

5 KRIOS – THE MISSING SYSTEM

Krios is a software system that incorporates the mechanisms and policies missing from the state of the art orchestration systems (such as Kubernetes [10]) to equip them to support LEO zones and thus enable a LEO compute platform. Leveraging established orchestrators ensures that the LEO compute cloud incorporates well-vetted, standardized components from existing, mature software stacks to avoid reinventing the wheel.

LEO constellation operators use Krios to enable deployment of unmodified containerized applications (LEO compute applications) in their constellations. These operators just need to specify their satellites' orbital information (through Two Line Elements (TLEs) [8]). Doing so enables LEO operators to offer a familiar interface, similar to terrestrial edge/-cloud systems, for deploying LEO applications (like CDNs, IoT aggregators, video calling applications, etc.). Application providers only need to add two new labels while deploying their application on Krios (Figure 5). The workflow for application deployment on LEO compute using Krios is summarized in Figure 6. While this paper focuses on containerized applications, the design principles introduced are also applicable for other paradigms of application deployment.

Unlike terrestrial deployments, Krios supports the additional capability to specify the target geographic zones – LEO zones, and to ensure that applications remain available in those zones despite the motion of satellites. This provides benefits of reduced bandwidth usage and latency compared to using terrestrial orchestration stacks. Krios achieves this by determining suitable satellites for a particular LEO zone and *proactively hands over* the application from them to the next set of suitable satellites. This ensures that the application is always directly accessible to the end users in the specified LEO zones, thus maintaining geostationarity of the application with respect to end users in the LEO zones.

Krios is built on top of an existing orchestrator such as Kubernetes inheriting the rich state-of-the-art features out of the box. These include replicasets that allow running multiple instances of the same application and autoscaler deployments that define thresholds to scale up or down application instances. Krios only modifies the scheduler and introduces the LEO controller that tracks all scheduled applications. Thus, applying the Krios design principles to different deployments

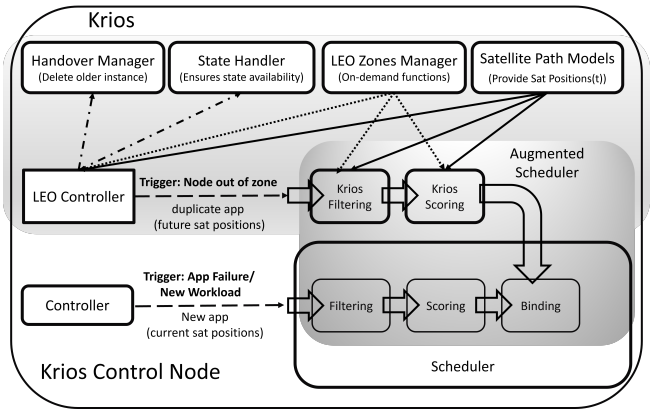


Figure 7: Krios augmented terrestrial orchestration stack.

can be done by a few minor engineering updates to the deployment controllers.

Figure 7 shows the capabilities of current orchestrators which are augmented by new capabilities designed as part of Krios. In the remainder of this section, we describe the design choices we make in Krios and discuss how they are encapsulated by the different Krios components that help Krios perform satellite mobility-aware orchestration to ensure continuous LEO compute application availability.

5.1 LEO Zones Orchestration

The first step in deploying an application is to provide a way for applications to specify a LEO zone where the application needs to be available. Krios chooses the least disruptive way to do so, by requiring application providers to specify the LEO zone as part of the application deployment request. Coupling LEO zones and applications together allows developers to define their own zone without any additional overheads.

Supporting software-defined LEO zones increases the number of zones the LEO controller needs to track. For every zone added as part of a new application, the scheduler and the LEO controller need to determine which satellites are inside the space projected region. On the contrary, if there were a fixed number of zones, the computation could potentially be shared among all the applications using the same LEO zone. However, as we show later in this section, the design of Krios design ensures that the overhead of supporting a LEO zone is fairly negligible and would not impact the performance of Krios even as the number of zones (applications) increases.

Once provided to a LEO orchestrator, the orchestration design must incorporate a LEO zone to decide where (on which satellite(s)) the application should be deployed. Terrestrially, a zone informs the orchestrator about the set of static nodes it can consider while making any deployment decision. Since the satellite nodes are constantly moving, the approach simply does not work for a LEO compute cloud. An intuitive alternative is to maintain a time-varying data structure for all zones that contains the list of satellites (nodes) that are currently available in the zone. However, this data structure must be constantly updated with the status of the nodes' availability from a zone, as a satellite could be accessible from a zone for a few seconds to a few minutes depending on its orbital characteristics [30]. Therefore, with no uniformity in satellite availability patterns, maintaining the list of satellites available in a LEO zone requires continuously computing the position of all the satellites in a loop to determine which satellites belong to a given zone. In addition to maintaining this list of nodes, the orchestrator also needs to identify which satellite is the best choice for deploying an application at a given time, based on the location of the satellite and the LEO zone. This requires continuously running another set of computations across all the satellites accessible from a zone. A terrestrial orchestrator does not need to perform any such computations as its node membership changes (barring failures or hotplugs) are rare and hence not considered as part of the design.

Instead, Krios incorporates the LEO Zones manager which represents a zone using a set of functions that are evaluated on demand to generate the corresponding zone state. The LEO Zones manager consists of three functions invoked by the different components of the orchestrator:

1. 'node_leaving_zone' invoked by the LEO controller to determine when a node is leaving the LEO zone;
2. 'get_zone_nodes' called during the filtering stage by the scheduler to get the list of nodes in the LEO zone; and
3. 'get_best_node' called during the scoring stage by the scheduler to find the best node to deploy the application.

Doing so, Krios only invokes functions in a on-demand fashion as opposed to continuous computations. Krios runs all the three functions once every handover cycle, which occur on average once every 3 minutes. This is in contrast to what would be required to maintain a time-varying data structure for all zones and all satellites. That would entail running the 'get_zone_nodes' and the 'get_best_node' functions for every scheduler tick for every LEO zone, or at least each time a node moves to a new zone. Note that nodes leave zones at a higher frequency than the frequency at which handovers need to be performed as Krios heavily prefers scheduling applications on nodes that are accessible to the LEO zone for a longer period.

LEO Zones Shape. Our current implementation considers circular LEO zones to simplify the calculations using the symmetrical structure of circles. A circular representation is a natural fit for satellite communications since the satellites that can serve a user forms a conical representation, which when projected results in a circle. As we show in Appendix B, this design is amenable to different shapes as well.

LEO Zones Size. Finally, the size of a LEO zone has a very crucial and unintuitive implication to how the space projected regions behave. The projected region for a LEO zone is the

LEO zone (km)	0	25	50	75	100	125
Projected Region (km)	1022	995	968	940	913	886

Table 1: Radius for LEO zone and its space projected region.

intersection of the projected regions of all points in it. Therefore, as the size of the LEO zone increases, the size of the projected region reduces which in turn increases the frequency of handovers. We show the geometrical calculations for determining the size of the space projected region in Appendix A. Table 1 shows the inverse relation between the radius of the LEO zone and its projected region for different sizes. As the size of the projected region decreases, so does the time between handovers. Therefore, Krios restricts the size of a LEO zones to a maximum 100 km radius so that there are more satellites available to service the region, and every satellite can service the region for at least 3 minutes. This size limit is large enough to be able to support larger cities within a single LEO zone (A 100 km radius or 31,000 km^2 area).

5.2 Krios Application Handovers

When a satellite running LEO compute application moves out of the projected region of its LEO zone, the application is no longer directly accessible to the end users in the zone. This is unlike a terrestrial edge/cloud, where the controller only focuses on failures and load spikes to maintain high availability. Instead, the LEO controller requires a mechanism to migrate applications from one satellite to another to ensure continuous availability of the application *in the LEO zone*. We refer to this migration as *application handovers*.

An approach to perform these application handovers could be using the cold-standby approach which entails having multiple 'cold' application instances that can be booted when required. However, this would require these instances running on at least 475 satellites (Figure 2) resulting in high resource utilization as well as increased overhead for managing these applications. Krios instead monitors the satellite running the application using the LEO controller, and initiates a handover when the satellite is about to move out of the LEO zone. This handover, driven by the handover manager entails initializing a new instance of the application on the new satellite and only deleting the older instance after the new instance is in the running state.

Further, in LEO compute, the availability of an application from the perspective of the orchestrator (determined by heartbeats) may not align with its availability from the LEO zone. As a result, it is conceivable that an application could be running on a satellite inaccessible to the LEO zone, but reachable from the orchestrator (through GSL or ISL hops) within a latency bound. This distinction is fundamentally different from terrestrial orchestrators, where availability in relation to the orchestrator is sufficient, as the position of the nodes isn't a concern. Therefore, the LEO controller determines when a handover is needed based on when the satellite hosting the application leaves the space projected region of the LEO zone using the 'node_leaving_zone' function exposed by the LEO Zones manager. This creates another set of independent, time-sensitive triggers that are driven by the satellite motion. We use satellite path models (SGP4 [53]) to determine when a particular satellite is exiting the LEO zone of an application and thus, triggering the handover to ensure availability and thus, *geostationarity* of the application.

When are Handovers Triggered? The LEO controller tracks all newly scheduled application instances. It creates a new thread that will find when the satellite is leaving the LEO zone using the 'node_leaving_zone' method. This thread sleeps until this trigger is activated and initiates the handover process.

However, the application handovers in LEO compute differ from the handovers in the cellular context as the entire application (and its associated state) is migrated instead of just the network connection and state. The initialization of the new instance of an application takes a non-trivial amount of time since it involves multiple steps such as pulling the application image, resource allocation, health checks on startup, etc. For instance, a k8s pod takes 5 seconds to initialize[97, 98][2]. While this initialization time can be minimized, it cannot be eliminated; and would always result in a downtime for the application every time a handover is triggered, i.e., every 3 minutes on average. Hence, if this handover trigger is reactive, i.e., the handover is triggered at the instant when a satellite is about to move out of the space projection region, the application will not be accessible during the initialization of the newer instance. Therefore, these handovers need to be done proactively to hide this delay. Krios estimates the additional time related to initialization, application transfer, etc., to perform *proactive, just-ahead-of-time handovers* to maximize uptime. The amount of time to be masked is calculated as

$$\triangle t = \triangle t_e + t_i + t_{AT} \quad (1)$$

where $\triangle t_e$ is the time error due to the inaccuracy of path model, t_i is the time to initialize the application on the new node, and t_{AT} is the time needed to move the application image/state (if needed). For the specific path model (SGP4) and orchestration stack (Kubernetes) we use, $\triangle t_e$ is about 1-3 km/day [99] which translates to about 4.6us, t_i is 5 seconds [97, 98], and t_{AT} depends on the size of the application image. For the specific applications we use in §7, t_{AT} is 0.2-1 seconds. For the specific path models and orchestrator we use, t_i and t_{AT} play a bigger role compared to t_e.

[2]Note that these benchmarks correspond to the orchestrator and worker node being collocated. In our case, there is added latency between the orchestrator and worker node that exacerbates this time.

How are Handovers Performed? Krios uses the handover manager to perform the handover process. The handover manager is triggered by the LEO controller when the application is about to be unavailable. A key requirement is that the orchestrator is able to schedule a new (originally idle) instance even when the current healthy instance is running (and serving user traffic without being overloaded). The handover manager initiates a new application instance, monitors its status and only once the new application instance is in the 'Ready' state, the handover manager kills the older application instance.

The handover process described so far is necessary even if the application image (and its associated state) is readily available on the desired satellite node. In certain cases, the application image and its state may need to be migrated as part of the handover process. Further, this process can benefit from incorporating the rich body of prior work including application container checkpoint/restart (CRIU) [100] or live migration techniques that have been used for application migration in other contexts [89, 91], but are overall expected to introduce higher data transfer requirements. We leave further optimizations to the handover process to future work.

Where to Schedule the Handover Instance? Terrestrially, the scheduler primarily focuses on resource availability while making a scheduling decision. The scheduling pipeline consists of multiple stages, but the ones of interest to us are the filtering stage and the scoring stage. The filtering stage filters out the nodes that do not satisfy the resource requirements of the application. The scoring stage then ranks the nodes based on the scoring function which is a combination of multiple metrics such as resource availability, node health, etc. The scheduler then selects the node with the highest score to deploy the application.

Filtering Stage. For LEO compute, the current filtering stage functions are insufficient as they do not take into account the position of the satellite with respect to the LEO zone of the application, which could lead to the application being deployed on a satellite that is currently not accessible from the LEO zone (Figure 9a). Therefore, Krios augments the filtering stage to filter nodes based on their presence in the space projected region of the LEO zone, in addition to the default resource availability requirements. This is done by computing the position of the satellites using satellite path models at the current time and evaluating whether the satellite lies within the space projected region (using the 'get_zone_-nodes' function exposed by the LEO Zones manager). This allows Krios to filter out nodes that may satisfy the resource availability requirements but are located farther away from the LEO zone.

Scoring Stage. Similarly, Krios augments the scoring stage to prioritize a node that will remain accessible to the LEO zone for the longest period of time. This is in contrast to other approaches, such as in the cellular context, where the user is generally handed over to the base station that is closest. Kubernetes, on the other hand, uses a random selection on nodes that satisfy the same affinity and priority requirements. Krios requires an additional scoring mechanism to ensure that a selected node can serve the application for the longest period of time without requiring frequent (and bandwidth-consuming) handovers. For circular regions, the symmetrical structure allows us to use the dot product between the velocity vector and the position vector of the satellite with respect to the center of the LEO zone. This ensures that satellites that are the farthest (but inside the space projected region of the LEO zone) and moving towards the center of the LEO zone are assigned a higher score and thus, help reduce the number of handoffs. While such a heuristic would not work for arbitrary shapes, that can be handled by projecting both the space projected region and the satellite trajectory on Earth to calculate how long the satellite will be accessible, as done in [21]. However, as the number of applications and LEO zones increases, it will require more sophisticated policies and better tuning of some of the parameters in Krios, that are part of future work.

5.3 Application State Management

A lot of prior work has advocated deploying stateless applications (or applications having an immutable state) on LEO satellites such as stateless cellular functions [70, 71], stateless video conferencing application [27, 85] and stateless IoT aggregation [84] (handling satellite motion by replicating the application on all the satellites). In such cases, Krios can be used to orchestrate the applications and ensure the same benefits with 400x lesser application instances.

However, other prior work has discussed potential performance benefits achieved by state transfer for CDNs [84] and web caching [28]. State transfer for LEO compute applications is enabled by the high bandwidth values for both the ISLs (up to 200 Gbps) and the GSLs (18-23 Gbps). Even applications with large sizes can be migrated during the 5-second handover process, which occurs every 2-3 minutes. It is important to note that these applications are such that their state can be rebuilt, therefore state transfer only impacts their performance but not their correctness.

An implicit assumption in terrestrial application deployments and orchestrators is the availability of networked storage (volumes) across the nodes of a cluster. This assumption is not valid for LEO compute as the physical presence of the networked storage on a satellite node would significantly impact the access latency – accessing storage present on another satellite or ground station will lead to increased bandwidth consumption and latency, thus negating the benefits of a LEO

compute cloud. Therefore, the storage for applications running on a satellite need to reside on the same satellite which can restrict the applications suitable for a LEO compute cloud.

To address the second class of applications, Krios uses an explicit state handler as an additional orchestration component for these applications. Krios provides a Krios-volume construct to store the state of the application, as is done in most edge/cloud settings. The state handler ensures that this Krios-volume is locally available to the application instance, even as the application gets handed over to other nodes during its lifetime. Krios implements state handovers proactively – similar to application handovers. While this may lead to some performance hit, e.g., the new instance not having the new cache entries added during the time of handover, this time is very small – just a few seconds. This can be addressed by leveraging techniques from extensive prior work on virtual storage migration [32, 34, 51, 72].

5.4 Krios Implementation

The current prototype implementation of Krios is built on top of Kubernetes and treats a pod definition as a LEO compute application. It uses the configuration parameters of all the satellites as two-line elements (TLEs) to predict the position and velocity of a satellite in the True Equator Mean Equinox frame. Krios is implemented as a Python package. Off-the-shelf, it can support the SGP4 path model and Kubernetes. The entire Python package consists of about 1000 lines of code. Our implementation of Krios prototype is available at https://github.com/GTkernel/krios.

6 EVALUATION METHODOLOGY

The goal of the evaluation of Krios is to establish that Krios is effective in providing applications the benefits they expect when deployed in a LEO compute cloud, compared to other possible deployment strategies for LEO compute cloud, and compared to other approaches used in state-of-the-art orchestration systems. Specifically, we ask the following questions:

- How well does Krios provide LEO compute benefits to applications (§7.1)?
- How effective are the mechanisms and policies introduced by Krios compared to current orchestration systems (§7.2)?
- What impact does Krios have operationally with varying conditions (§7.3)?

Experimental Methodology. To the best of our knowledge, there is no LEO compute cloud orchestration system to compare as a baseline. Hence we compare our results against Kubernetes to show the gaps of current off-the-shelf orchestration stacks. We also create incremental baselines augmenting stock Kubernetes to show the impact of the individual Krios components. We run all our experiments for 100 minutes (just over the orbit time of 96 minutes) to show the validity over an entire orbit duration. We repeat the experiments multiple times to show the consistency of our results.

Emulation Setup Implementation. Existing emulation platforms Celestial [85] and StarryNet [68] are designed to demonstrate the feasibility and the benefits of LEO compute by replicating the applications on all satellites. Celestial deploys all the satellites and ground stations using ephemeral microVMs while StarryNet does so using containers. Since both these emulators run applications on all satellites, it resulted in the orchestration of the emulator being tightly coupled with the orchestration of the application. Hence, it would have taken a significant engineering effort to specifically replace the LEO compute orchestration layer separately. Further, deploying Kubernetes on microVMs and containers required considerable engineering effort. Combining these two changes made it harder for us to extend these systems leading us to build our custom emulation framework. However, it is important to note that we borrowed the logic for managing satellite positions and modifying the GSL/ISL latencies (essentially all the computations related to satellite motion) while building our stack on top of virtual machines (VMs). Our emulator is also available as part of the Krios git repo.

To effectively answer the questions we pose in our evaluation, we need to emulate the deployment of an application from the perspective of a particular LEO zone. While we do need to model the position of all the satellites in the constellation, we primarily need to focus on the location (which also determines the latency for the end user) of the satellite node running the application, and the node to which the application needs to be handed over to. To maintain the fidelity of this emulation, the latencies between the end user and these two satellite nodes, as well as the orchestrator and these two satellite nodes, need to be modified to reflect the position of the satellites. Further, we are only focused on studying the latency characteristics based on the position of the satellite with respect to the LEO zone, hence we do not worry about any queueing artifacts that may be introduced due to satellite hops. While two satellite nodes can help us evaluate for a single handover, i.e., to run the emulation for about 3 minutes, to run a 100 minute emulation will require larger number of nodes. Instead, we use a virtualized emulation setup where we constantly change the identities of the satellite nodes emulated by the VMs, so that we can reuse them for multiple handovers. To support one application, we need one auxiliary satellite node onto which a running application instance can be handed over to. Essentially, our emulator needs two satellite VMs per application and another two as orchestrator and client. For all our evaluations, we use 6 VMS: four to emulate satellite nodes, one orchestrator node, and one user terminal node. Note that this set up emulates four satellites at a given time instance. Over the course of 100 minutes, these VMs claim the identity of ~300 satellites. This allows us to

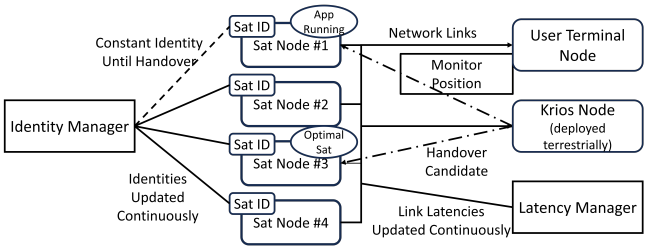


Figure 8: Architecture of our emulation setup for Krios.

run the emulation for 100 minutes with fewer nodes while maintaining the fidelity of our experiments.

The emulator, shown in Figure 8, operates by assigning to the satellite nodes identities of satellites that are accessible from the LEO zone being used for evaluation. One node (VM) emulates the satellite where the application is deployed, and executes it. The identities of the remaining 3 satellite nodes are updated every second, while ensuring that one of them is always assigned the identity of the optimal satellite at that instance. This is feasible because of the forward-predictability in satellite motion and a backward-knowledge about the application being emulated. In a single application case, the other two are assigned identities of random satellites accessible from the LEO zone. The identity manager does not change the identity of the satellite running an application until the application is handed over to another satellite to maintain correctness of the emulation. The emulator also continuously modifies the RTT between the satellites and the orchestrator, and between the satellites and the client, using the Python tcconfig package [52]. Each satellite node (24 cores, 2.67 GHz) is similar to the Hewlett Packard Enterprise EL 8000s (24 cores, 2.4GHZ) used by OrbitsEdge in satellites to provide an edge offering [5, 83]. We run our Krios implementation on the orchestrator node.

Emulated Constellation Characteristics. We evaluate Krios by emulating two different constellations - (1) first shell of Starlink, and (2) all the 6416 Starlink satellites (as of October 2024). The first shell consists of 1584 satellites split evenly across 72 orbital planes (22 per orbital plane). We assume a grid of ground stations uniformly spread on the EarthâĂŹs surface similar to [48]. We use SGP4 to predict the position of every satellite (similar to [28, 41, 65]) every second and compute the latency from the user terminal/ground station to satellites using ground stations as relays [48], using each of the ground-satellite link RTT as 9ms (18 ms [63] for ground-satellite-ground and back). While prior work [27, 30, 47, 48, 50, 65] has calculated RTT as the distance covered divided by the speed of light (about ~4 ms), we use 9 ms as it is the lowest reported latency by Starlink. For the latency along the ISLs, we use the propagation delay value as there are no such measurements available. For our latency measurements, we do not introduce any artificial loss to explicitly focus on the delay due to the network.

Latency (ms)	Smokeys		Oil Rig		Instances
%ile	95	99	95	99	per App
Ground Station edge	1600	1928	37.8	37.6	N/A
Kubernetes	13780	18746	667	667.2	1
Naive LEO compute	1144	1164	19.2	19.5	475
Krios LEO compute	1144	1164	19.2	19.3	1.03
Comparison	12x	16x	34.7x	34.5	461x

Table 2: LEO compute workloads latency performance with Krios compared to different deployment scenarios. Comparison refers to gains over Kubernetes for latency and Naive edge for instances.

Baseline	GS edge	k8s	Naive	Krios
Data-plane bytes/day	760GB	15TB	380GB	380GB
Control-plane bytes/day	N/A	116Mb	55GB	50GB

Table 3: Data-plane and control-plane bytes/day for serving trail map at Smokey Mountains.

Req Size	Resp Size	GS (0)	k8s (1)	naive (475)
Small	Small	2x	38x	1x
Small	Large	1.9x	27x	1x
Large	Small	2.3x	44x	1x
Large	Large	1.9x	40x	1x

Table 4: Benchmarking 99th percentile latency of templatized LEO compute workloads using Krios compared to other deployment strategies. The numbers in brackets correspond to the number of instances needed. Krios only requires 1.03 instances

Evaluation Metrics. We look at the worst-case latency to reach the application from the LEO zone. We use this metric to show the effectiveness of Krios-enabled LEO compute for two applications, as well as to compare the effectiveness of Krios against the incremental baselines. Using microbenchmarks, we demonstrate the effectiveness of individual mechanisms and policies to characterize the performance and limitations of Krios.

7 EVALUATION

7.1 LEO Compute Application Benefits with Krios

We present results for two concrete applications: (i) a web serving application in a remote national park and (ii) an IoT aggregator collecting sensor information from oil rigs in the Gulf of Mexico. These applications require targeted deployments to serve specific geographical ares and they are representative of LEO compute applications that are amenable for today's LEO satellite use cases – web access in remote areas and LEO satellite driven IoT aggregation. Both these applications do not have external mutable state requirements. For the web access application, we use fixed state which rarely changes. The IoT aggregator only has to track instantaneous availability of the sensors. We also look at templatized applications modeled after standard edge/cloud applications

varying the request and response sizes. We compare the performance of a Krios-enabled LEO compute cloud with a naive LEO compute with 475 instances, and a Kubernetes driven LEO compute deployment with 1 instance, while also drawing comparisons with using an edge deployed on ground stations[3]. The Kubernetes driven baseline treats all satellite nodes equally, due to its location agnostic nature.

Serving Trails at Smokey Mountains. This application serves the trail map (~100 MB) at the Great Smokey Mountains national park. This is one of the most popular national parks in the US, but has very poor cellular network connectivity, making it an ideal fit for LEO satellite based internet service. This application requires a 50 km radius LEO zone to cover the entire national park.

Table 2 shows the latency percentiles for accessing this static trail map in the different scenarios. Given the size of the map size, multiple packets are needed to deliver the entire map. This needs to comply with the TCP window and hence the latency increases even more if the satellite is not directly accessible. While the naive LEO compute gets similar performance as Krios, it does so with 475 application instances, with each of them only serving for a maximum of 3-4 minutes per 97 minutes – cost that cannot be justified for a targeted application like this. Instead, Krios achieves the same benefit from the perspective of application responsiveness, with just ~1 instance. Krios-enabled LEO compute cloud requires the application image to be transferred to the incoming satellite during handovers. We found that Krios performs a handover every 200 seconds (median) for this LEO zone, resulting in approximately 432 handovers in a day. In our implementation, we packaged this trail map inside an Nginx container resulting in an image size of 116 MB, which results in a total of 50 GB spent on performing the handovers (Table 3). This is still lower than the bandwidth consumed by 475 application instances (475 * 116 MB = 55 GB) in a naive LEO compute deployment as the application images need to be transferred to the satellites at some point. Krios-enabled LEO compute cloud performs better compared to the ground station edge deployment in both latency and overall bytes on the network, even after considering the bytes spent in handovers. The Kubernetes baseline performs even worse as the satellite will be quite far away most of the times.

This park had an annual attendance of 14 million in 2021 which translates to approximately 38000 visitors a day. Therefore, even if just 10% of the visitors access this map in a day, it would result in $3800 \times 100 * 2$ MB i.e. 760 GB of data being sent over the GSLs in a bent-pipe of a ground station edge deployment. On the other hand, if deployed through Krios-enabled LEO compute, half of these bytes will be saved, achieving latency and bandwidth usage reduction while practically enabling a LEO compute.

Monitoring Oil Rigs in the Gulf of Mexico. We show the applicability of Krios to two oil rig monitoring applications. We chose two oil rig locations [81] that are 135 km away and used a 5 km radius LEO zone. In this case, the oil rig sensors constantly ping heartbeat packets (4B) to the aggregator. Table 2 shows the latency performance in this scenario.

Templatized LEO Compute Application Performance. We present the latency behavior of templatized LEO compute applications when they are deployed on LEO compute using the different deployment strategies in Table 4 as a multiplier to the latency observed by Krios. These applications are implemented on top of Nginx containers modified to these request/response sizes. The bandwidth behavior for these applications will be the same, exhibiting similar patterns as shown in Table 3 as bytes on the network are always proportionally impacted based on how many handovers are needed (for Krios), how many satellites the application image needs to be transferred (for naive LEO compute), and how many hops it takes to reach the application (for k8s driven edge).

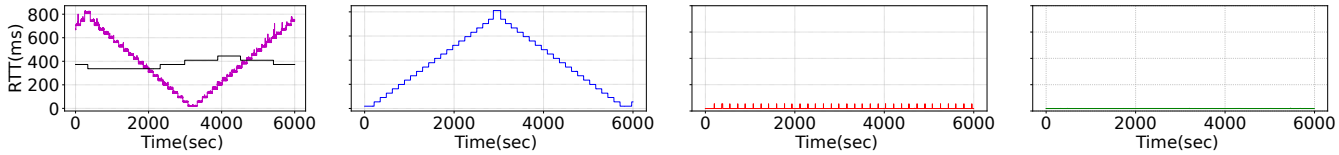
In summary, these results demonstrate that Krios enables a practical LEO compute cloud and drastically reduces the satellite resource and link consumption.

7.2 Effectiveness of Krios

The previous results show the benefit of using a LEO compute orchestrator to manage application deployment over a naive deployment where an application is ubiquitously present a priori. Next, we justify the design decisions in Krios by showcasing their contributions to overall system effectiveness while using terrestrial orchestrators and policies for handovers as baselines. For these experiments, we assume the presence of the application images on the satellites to specifically measure the impact of individual mechanisms and policies.

Benefits of Krios Mechanisms. To show the incremental benefits of Krios mechanisms and components, we measured the worst-case latency to access the satellite from a LEO zone. As this increase in latency is only due to the increase in the number of hops to reach the satellite, it also results in greater bandwidth utilization. To study this, we used our emulation setup with the first Starlink shell (1584 satellites) and measured the latency from the perspective of a LEO zone centered in San Francisco (37.7749^oN, 122.4194^oW) with a radius of 100 km. We use the stateless Nginx service with a request size of 64 Bytes and a response size 615 Bytes. Figure 9 shows the latencies observed by the clients while accessing the Nginx service. We compare four scenarios ranging from path-unaware stock Kubernetes to Krios while adding in the Krios scheduler and (not proactive) application handovers as the intermediate baselines.

[3]This can also be considered as an optimized bent-pipe.



(a) Stock k8s (b) Krios scheduler (c) Krios LEO Controller (d) Krios

Figure 9: Comparing the worst-case user latencies to reach an application deployed by Stock k8s, Stock k8s with Krios scheduler, stock k8s with Krios scheduler and LEO controller, and Krios. All four figures have the same y-axis

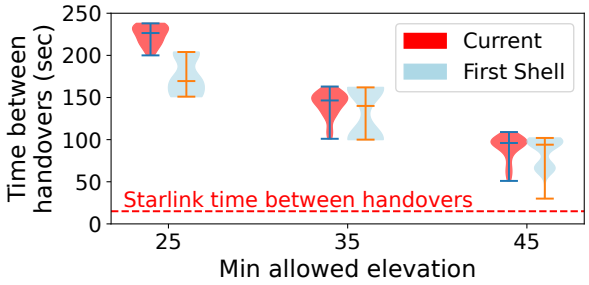


Figure 10: Comparing time between handovers for different minimum elevation angle constraints using Krios. Starlink currently uses a constant 15 second handover interval.

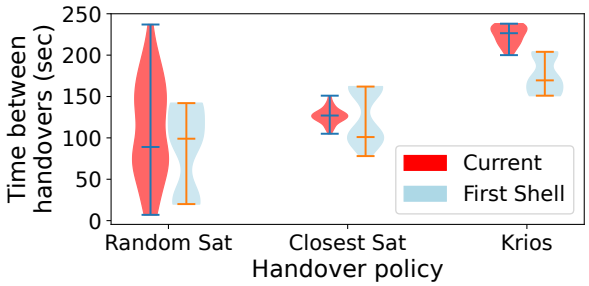


Figure 11: Time between handovers for Krios scoring vs other wireless policies.

For the baseline k8s deployment, we consider the entire constellation as a single cluster with location-unaware orchestration. Stock Kubernetes (Figure 9a) will result in high variations in the latencies observed by the users as any satellite could be chosen from the constellation and its position will keep varying with time. We show two different examples of satellites that could be chosen by such an orchestrator resulting in varying access latencies. While adding Krios scheduler provides direct access at the beginning of the emulation resulting in lower latencies (Figure 9b), once the satellite moves out of range, the latency increases. In both of these scenarios, it is important to observe that if the latency is greater than the median latency offered by Starlink 42ms [56, 61], a LEO compute cloud starts losing its benefits and actually becomes detrimental to the entire system.

This is greatly improved by adding Krios LEO controller that can continuously handover applications (Figure 9c). However, if the handovers aren't proactive, i.e., they are triggered when the satellite is about to be inaccessible, it creates multiple spikes during those times while the new application instance is being initialized. While these seem like small latency spikes, they still result in application downtimes during the spikes. Such downtimes in observation satellites would imply the LEO compute application tuned to cover one geographical region is running over another region that may have completely different characteristics and may require a different model. To resolve this, Krios performs proactive handovers masking the time required for the initialization of the new instance. This allows Krios to consistently achieve direct access (Figure 9d) and thus maximize the latency and bandwidth savings.

Benefits of Krios-handovers compared to Periodic Handovers. We compare the proactive handovers triggered by Krios to the periodic handovers by Starlink today. Recent measurement studies [57, 78, 95] for the Starlink constellation indicate handovers occurring every 15 seconds, with increased elevation angle shown as one of the key benefits achieved – more than 80% of the satellites chosen had an elevation angle of at least 45^o [95]. We measure how Krios would perform when the elevation angle is more constrained as it will obviously result in more frequent handovers in Figure 10. As expected the time between handovers reduces (and thus the number of handovers increases) as the minimum allowed elevation angle is increased. However, even at 45^o, the median time between handovers is 94 seconds (6x of the Starlink frequency). Essentially, Krios enables tuning the quality of the network links used while still outperforming periodic handovers aimed at similar benefits.

Benefits of Krios Scoring Policy. We compare the path-aware node selection policy used by Krios during handovers with the random node and closest node selection policies. Figure 11 shows the comparison of the average time between handovers of each of these policies. As expected, a random node selection policy requires very frequent handovers and has high variations since this policy will select nodes visible for shorter durations with the same probability as nodes visible for a longer duration. While the closest node selection policy improves over the random policy, a node is used for just about half the time a satellite is accessible from the zone because the closest node would have covered half the distance in the LEO zone's projected region and hence is available for

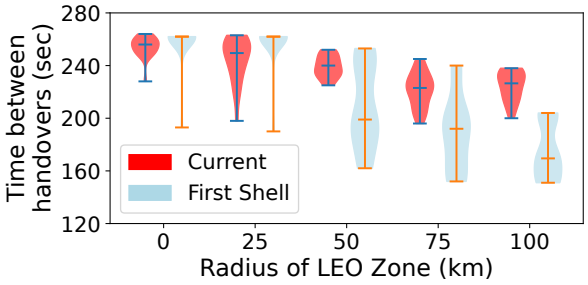


Figure 12: Comparing the time between handovers for LEO zones with varying sizes.

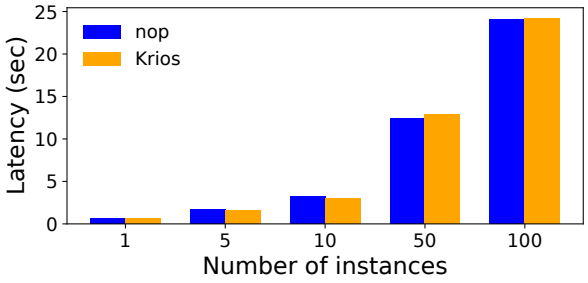


Figure 13: Comparing the scalability of the Krios-scheduler with a nop-scheduler.

just half the time. The Krios scoring policy performs 1.5× better than the closest node selection policy and 2× better compared to the random node selection policy by scheduling on nodes that will be accessible for the longest duration amongst the alternatives available.

In summary, the above results demonstrate that Krios effectively eliminates the impact of LEO satellite mobility on application performance and improves the current policies used for handovers, optimizing the use of LEO compute.

7.3 Krios Microbenchmarks

We next evaluate several aspects of Krios that were not explicitly covered in earlier sections.

Impact of LEO zones size. Figure 12 shows the impact of the size of a LEO zone on the time between handovers. We compare the time between handovers for circular LEO zones with radius 0km, 25km, 50km, 75km, and 100km. As the size of the LEO zone increases, the time between handovers reduces leading to an increased number of handovers. This downward trend is consistent with our analysis in §5.

Krios Scheduler Scalability. Figure 13 shows the time taken by the Krios scheduler to schedule different number of applications, each with a different LEO zone. As we are trying to benchmark our scheduler, we only measure the time taken by the scheduler to make a scheduling decision and disregard the time taken by the rest of the control plane to deploy the application. Concretely, we measure the time taken by an application to reach the 'Scheduled' state in Kubernetes. We compare the Krios scheduler implementation with a no-op scheduler which simply selects a random node for scheduling and plot the average scheduling time for scheduling one pod. We perform these measurements multiple times and present the average values. We observe that both the schedulers have near-identical performance. Concretely, Krios adds ~3-5% scheduling overhead. This is because the additional work introduced by Krios, in the filtering and the scoring stages, remains a small fraction of the underlying scheduler execution. Important to note here is that any handovers that need to be performed are effectively new applications to be scheduled by Krios. So, from the orchestrator's perspective, the number of apps to be scheduled (on x-axis) is either all new deployments or all handovers or a mix of both. This only shows that the Krios scheduler decision remains as scalable as the underlying terrestrial orchestrator (Kubernetes in our case) scheduling loop.

	CPU (millicores)	Memory (bytes)
Kubernetes	2020	4254Mi
Krios	2469	4594Mi

Table 5: CPU and memory utilization k8s vs. Krios.

# Pods	10	50	100
Creation	2.3	8.57	21
Deletion	2.36	10	19

Table 6: Additional network bytes by Krios for handovers.

Satellite Compute Overhead. On the LEO satellites, the compute overhead by Krios is experienced at the time of application handovers when there will be two instances running concurrently. However, this overhead is observed for just 5 seconds (Equation 1) every 3 minutes (Figure 11), i.e., for about 3% of the time. To be able to provide similar availability guarantees without Krios, 475 satellites need to concurrently run the same application instance (Figure 2). Thus, instead of running 475 instances all the time, Krios runs two instances for very short durations frequently, while running a single instance for 97% of the time.

Krios Control Node Compute Overhead. Table 5 shows the additional compute overhead introduced by Krios on the control node. However, this increase is acceptable since the control node is deployed terrestrially, e.g., in ground stations where this increase can be accommodated.

Krios Control Message Overhead. The LEO controller module of Krios creates the application instance on a different node (directly accessible from the LEO zones) and deletes the instance from the earlier node. Table 6 shows the number of bytes exchanged between the ground station (control node) and the satellite for pod creation and pod deletion in addition to the bytes required to transfer the application image. Using these numbers, we estimate that per pod, Krios causes about 400 bytes (~200 bytes each for creation and deletion)

of network overhead during every handover action (which happens once every 3 minutes). While this does add some extra bytes to the satellite network, it is considerably lesser than the number of bytes it helps reduce on the satellite network.

In summary, Krios enables LEO compute while incurring minimal overheads on control plane nodes, and negligible overhead on the LEO satellites as well as the satellite links.

8 DISCUSSION

Energy Utilization. LEO satellites harvest solar energy for carrying out their regular functions as well as for enabling compute. In that sense, orchestration systems should model the energy harvested by a satellite as well as energy consumed by candidate applications while making scheduling decisions With Krios, we develop a capability for targeted deployment of LEO applications (closer to their desired end users). Future extensions of Krios can incoporate other metrics, including energy harvesting models and energy utilization models [29], as part of orchestration systems for the LEO compute cloud.

Suitability of LEO compute applications. An application deployed on LEO satellites will incur overheads either to deploy it on all satellites, like in prior work, or due to handovers, as is done by Krios. This creates a tradeoff that should be considered when determining how to deploy an application as part of the LEO compute cloud. Applications with small, immutable state, such as the selective forwarding units in a video calling applications, ML inference models, etc., can easily reside on satellites (given the large available storage), but other applications such as CDNs and IoT aggregators with large, mutable states would require a careful analysis of the benefits provided by deploying them on LEO satellites to ensure a positive tradeoff between the cost for updating the state and benefits gained from deploying on the satellites. However, as Krios is designed to be application-transparent, any targeted LEO compute deployment will benefit from even if all terrestrial edge/cloud applications may not be suitable for LEO compute deployment. We consider the analysis of suitability of applications to be deployed on satellites out of scope for this work.

Scaling applications. Krios highlights the fundamental mechanisms and policies required to orchestrate LEO compute applications. While these would remain valid even as the number of applications scale, it would require more sophisticated policies and potentially running multiple orchestrators. We leave the ensuing discussions on necessary policies and coordination mechanisms for future work.

Extending Software-defined LEO Zones. In this paper, we introduce the notion of software-defined LEO zones. This concept allows application providers to specify their requirements in a declarative manner that the system is able to satisfy. This is in contrast with how the terrestrial ecosystem has existed wherein the application provider needs to specify exactly where their application needs to run. We hypothesize that this concept can be easily adapted for the terrestrial edge/cloud scenario allowing providers greater freedom and helping the infrastructure better load balance.

9 RELATED WORK

The field of LEO satellite computing has seen a lot interest focusing on enabling different applications, both for communication and observation satellites. Orbital Edge Computing (OEC) [41] proposes an architecture where satellites share the computational load to perform inference on captured images. However, the orchestration in this case is tied to the application logic itself, making it harder for the system to support multiple applications. While Kodan [39] adds the ability to run multiple applications, the applications need to undergo a lot of preprocessing before they're deployed on the satellites. FedSpace [93] highlights the challenges associated with federated learning training on satellites while focusing on data staleness. [84] discusses the possibility of deploying a CDN on the LEO satellite infrastructure with the satellites acting as the points-of-presence (PoP). Similarly, [70] discusses the feasibility of deploying a cellular core on LEO satellites. But all of these involve running the same application on all the satellites. Krios, on the other hand, focusses on the general systems issues faced by LEO compute applications to present an orchestration stack for deploying general-purpose applications, not just for the use cases evaluated in this paper, but also for the applications mentioned above.

Bhattacherjee et al. [27] concretely discuss the possibility of deploying an edge on LEO satellites and discuss the potential challenges that need to be addressed for a successful LEO compute cloud. They also discuss the range of applications that could potentially benefit such as content distribution, multi-user gaming, etc. However, this work is largely a feasibility study and does not discuss the concrete challenges of deploying such applications. [87] assess the different organizational paradigms for applications (OPAs), namely VMs, containers, and serverless functions, in the context of LEO deployments. Krios goes beyond using path models to defining the abstractions, mechanisms and policies required for system support of a LEO compute orchestrator including support for application handovers.

Recent studies have also provided a range of network measurements data on Starlink satallite deployments [57, 78, 95] These have uncovered the occurrence of periodic handovers that govern which satellite a particular user terminal talks to. We also compare Krios handovers with periodic handovers in Figure 10 and observe that even in a more constrained setting, Krios reduces the frequency of handovers.

10 CONCLUSION

We presented Krios, a software system that makes it feasible for LEO satellite providers to offer an effective LEO compute cloud. Krios achieves this by centering its design around the novel *software-defined LEO zones* abstraction while incorporating necessary mechanisms and policies for *proactive and predictive application handovers*. Krios is designed to ensure its seamless integration with existing terrestrial orchestration stacks and to avoid disruptive changes to application development, thus, paving the way for a practical LEO compute cloud. We consider Krios as a starting point to show there are gaps in terrestrial software system designs that need to be addressed to develop software systems for LEO compute cloud, and perhaps in other extra-terrestrial settings [86].

ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd Valerio Schiovani for their constructive feedback that helped us improve this paper. We also thank James Choncholas and Ranjan Sarpangla Venkatesh for reviewing earlier versions of this paper. This project was partially supported by NSF grant CNS-2345827 as well as grants and gifts from Cisco and the Georgia Smart program.

REFERENCES

[1] 2018. A Platform for Computing at the Mobile Edge: Joint solution with HPE, Saguna and AWS. https://d1.awsstatic.com/whitepapers/platform-for-computing-at-the-mobile-edge-hpe-saguna-aws.pdf. [Accessed 10/October 7th,.

[2] 2018. Launch Costs to Low Earth Orbit, 1980-2100. https://www.futuretimeline.net/data-trends/6.htm. (Accessed on October 7th, 2024).

[3] 2018. LTE Handover Latency Calculation (Access Node). https://www.techplayon.com/lte-handover-latency-calculation-access-node/. (Accessed on October 7th, 2024).

[4] 2020. FCC Selected Application for Space Exploration Holdings, LLC. SESLIC20190906011171.

[5] 2020. One Giant Step Towards Launch. https://orbitsedge.com/blog/f/one-giant-step-towards-launch. (Accessed on October 7th, 2024).

[6] 2021. Google Wins Cloud Deal from Elon Musk's SpaceX for Starlink Internet Connectivity. https://www.cnbc.com/2021/05/13/google-cloud-wins-spacex-deal-for-starlink-internet-connectivity.html. (Accessed on October 7th, 2024).

[7] 2021. Planet Data Enables Climate Transparency Through New Partnership With Climate TRACE. https://www.planet.com/pulse/planet-data-enables-climate-transparency-through-new-partnership-with-climate-trace/. (Accessed on October 7th, 2024).

[8] 2022. NORAD Two-Line Element Set Format. https://celestrak.org/NORAD/documentation/tle-fmt.php. (Accessed on October 7th, 2024).

[9] 2023. Space: The ultimate testing ground for data storage technologies. https://www.techrepublic.com/article/space-testing-ground-data-storage-technologies/. (Accessed on October 7th, 2024).

[10] 2024. Kubernetes. https://kubernetes.io/. (Accessed on October 7th, 2024).

[11] 2024. Lacuna Space. https://lacuna.space/. (Accessed on October 7th, 2024).

[12] 2024. Lumen Orbit. https://www.lumenorbit.com/. (Accessed on October 7th, 2024).

[13] 2024. OneWeb. https://www.oneweb.world. (Accessed on October 7th, 2024).

[14] 2024. Project Kuiper. https://www.aboutamazon.com/what-we-do/devices-services/project-kuiper. (Accessed on October 7th, 2024).

[15] 2024. Starlink. https://www.starlink.com. (Accessed on October 7th, 2024).

[16] 2024. Telesat. https://www.telesat.com. (Accessed on October 7th, 2024).

[17] Aaron Leong. 2022. Why Starlink download speeds are experiencing double-digit dips? https://www.digitaltrends.com/computing/why-starlink-download-speeds-are-dropping-by-so-much. (Accessed on October 7th, 2024).

[18] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. 2017. Mobile edge computing: A survey. *IEEE Internet of Things Journal* 5, 1 (2017), 450–465.

[19] Abner Li. 2022. Android 14 will support satellite connectivity and partners. https://9to5google.com/2022/09/01/android-14-satellite-support. (Accessed on October 7th, 2024).

[20] Hawaiian Airlines. 2022. Hawaiian Airlines to Offer Free, High-Speed Starlink Internet Connectivity on Transpacific Fleet. https://newsroom.hawaiianairlines.com/releases/hawaiian-airlines-to-offer-free-high-speed-starlink-internet-connectivity-on-transpacific-fleet. (Accessed on October 7th, 2024).

[21] Irfan Ali, Naofal Al-Dhahir, and John E Hershey. 1999. Predicting the visibility of LEO satellites. *IEEE Transactions on Aerospace and Electronic Systems* 35, 4 (1999), 1183–1190.

[22] Angus Ledwidge. 2022. Starlink in disaster zones: How satellite technology was delivered to Australian flood victims. https://7news.com.au/weather/natural-disasters/starlink-in-disaster-zones-how-satellite-technology-was-delivered-to-australian-flood-victims-c-6102595. (Accessed on October 7th, 2024).

[23] Ari Bertenthal. 2022. Starlink Passes 4 Million Subscribers Globally. https://broadbandbreakfast.com/starlink-passes-4-million-subscribers-globally/. (Accessed on October 7th, 2024).

[24] Akash Badshah, Natalie Morris, and Matthew Monson. 2023. Over-The-Vacuum Update–Starlink's Approach for Reliably Upgrading Software on Thousands of Satellites. In *Proceedings of the AIAA/USU Conference on Small Satellites*.

[25] Panagiotis Barmpoutis, Periklis Papaioannou, Kosmas Dimitropoulos, and Nikos Grammalidis. 2020. A review on early forest fire detection systems using optical remote sensing. *Sensors* 20, 22 (2020), 6442.

[26] Ketan Bhardwaj, Ming-Wei Shih, Pragya Agarwal, Ada Gavrilovska, Taesoo Kim, and Karsten Schwan. 2016. Fast, Scalable and Secure Onloading of Edge Functions Using AirBox. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 14–27.

[27] Debopam Bhattacherjee, Simon Kassing, Melissa Licciardello, and Ankit Singla. 2020. In-orbit Computing: An Outlandish Thought Experiment?. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*. 197–204.

[28] Vaibhav Bhosale, Ketan Bhardwaj, and Ada Gavrilovska. 2020. Toward Loosely Coupled Orchestration for the LEO Satellite Edge. In *3rd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 20)*.

[29] Vaibhav Bhosale, Ketan Bhardwaj, and Ada Gavrilovska. 2023. Don't Let Your LEO Edge Fade at Night. In *1st Workshop on Hot Topics in System Infrastructure (HotInfra 23)*.

[30] Vaibhav Bhosale, Ahmed Saeed, Ketan Bhardwaj, and Ada Gavrilovska. 2023. A Characterization of Route Variability in LEO Satellite Networks. In *Passive and Active Measurement: 24th International Conference, PAM 2023, Virtual Event, March 21–23, 2023,*

Proceedings. Springer, 313–342.

[31] Nathaniel Bleier, Muhammad Husnain Mubarik, Gary R Swenson, and Rakesh Kumar. 2023. Space Microdatacenters. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 900–915.

[32] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. 2007. Live Wide-Area Migration of Virtual Machines Including Local Persistent State. In *Proceedings of the 3rd International Conference on Virtual Execution Environments* (San Diego, California, USA) *(VEE '07)*. Association for Computing Machinery, New York, NY, USA, 169–179. https://doi.org/10.1145/1254810.1254834

[33] Brian Wang. 2019. SpaceX Starlink Satellites Could Cost $250,000 Each and Falcon 9 Costs Less than $30 Million. https://www.nextbigfuture.com/2019/12/spacex-starlink-satellites-cost-well-below-500000-each-and-falcon-9-launches-less-than-30-million.html. (Accessed on October 7th, 2024).

[34] Emré Celebi, Tal Garfinkel, and Min Cai. 2011. The Design and Evolution of Live Storage Migration in VMware ESX. In *2011 USENIX Annual Technical Conference (USENIX ATC 11)*. USENIX Association, Portland, OR. https://www.usenix.org/conference/usenixatc11/design-and-evolution-live-storage-migration-vmware-esx

[35] Kejie Chen, Jean-Philippe Avouac, Saif Aati, Chris Milliner, Fu Zheng, and Chuang Shi. 2020. Cascading and pulse-like ruptures during the 2019 Ridgecrest earthquakes in the Eastern California Shear Zone. *Nature communications* 11, 1 (2020), 22.

[36] Chris Wedel. 2023. Starlink internet is going from rural savior to unreliable luxury. https://www.xda-developers.com/starlink-internet-rural-savior-unreliable-luxury/. (Accessed on October 7th, 2024).

[37] Potomac Officers Club. 2023. Planet Labs Completes Months-Long Inter-Satellite Link Test. https://potomacofficersclub.com/news/planet-labs-has-concluded-testing-for-its-inter-satellite-communications-link-hardware/. (Accessed on October 7th, 2024).

[38] Dan Swinhoe. 2021. SpaceX's Starlink to house satellite ground stations at Google data centers, partner on Edge. https://www.datacenterdynamics.com/en/news/spacexs-starlink-to-house-satellite-ground-stations-at-google-data-centers-partner-on-edge/. (Accessed on October 7th, 2024).

[39] Bradley Denby, Krishna Chintalapudi, Ranveer Chandra, Brandon Lucia, and Shadi Noghabi. 2023. Kodan: Addressing the Computational Bottleneck in Space. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 392–403.

[40] Bradley Denby and Brandon Lucia. 2019. Orbital Edge Computing: Machine Inference in Space. *IEEE Computer Architecture Letters* 18, 1 (2019), 59–62.

[41] Bradley Denby and Brandon Lucia. 2020. 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*. 939–954.

[42] Kiruthika Devaraj, Matt Ligon, Eric Blossom, Joseph Breu, Bryan Klofas, Kyle Colton, and Ryan Kingsbury. 2019. Planet high speed radio: Crossing Gbps from a 3U cubesat. In *Proceedings of the AIAA/USU Conference on Small Satellites*.

[43] Doug Mohney. 2020. Terabytes From Space: Satellite Imaging is Filling Data Centers. https://www.datacenterfrontier.com/internet-of-things/article/11429032/terabytes-from-space-satellite-imaging-is-filling-data-centers. (Accessed on October 7th, 2024).

[44] Edward Wyatt and Noam Cohen. 2014. Comcast and Netflix Reach Deal on Service. https://www.nytimes.com/2014/02/24/business/media/comcast-and-netflix-reach-a-streaming-agreement.html. (Accessed on October 7th, 2024).

[45] Emma Woollacott. 2022. Starlink Terminals Smuggled Into Iran - But How Effective Can They Be? https://www.orbes.com/sites/emmawoollacott/2022/10/25/starlink-terminals-smuggled-into-iranbut-how-effective-can-they-be/?sh=1c2952561027. (Accessed on October 7th, 2024).

[46] Warren Frick and Carlos Niederstrasser. 2018. Small Launch Vehicles-A 2018 State of the Industry Survey. In *Proceedings of the AIAA/USU Conference on Small Satellites*.

[47] Mark Handley. 2018. Delay is not an option: Low latency routing in space. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*. 85–91.

[48] Mark Handley. 2019. Using Ground Relays for Low-Latency Wide-Area Routing in Megaconstellations. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*. 125–132.

[49] Harry Menear. 2021. KDDI taps SpaceX's Starlink for rural cellular backhaul. https://mobile-magazine.com/wireless-networks/kddi-taps-spacexs-starlink-rural-cellular-backhaul. (Accessed on October 7th, 2024).

[50] Yannick Hauri, Debopam Bhattacherjee, Manuel Grossmann, and Ankit Singla. 2020. " Internet from Space" without Inter-satellite Links. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*. 205–211.

[51] Takahiro Hirofuchi, Hirotaka Ogawa, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. 2009. A Live Storage Migration Mechanism over WAN for Relocatable Virtual Machine Services on Clouds. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE, 460–465.

[52] Tsuyoshi Hombashi. 2016. Welcome to tcconfig's documentation! https://tcconfig.readthedocs.io/en/latest/. (Accessed on October 7th, 2024).

[53] Felix R. Hoots and Ronald L. Roehrich. 1980. Models For Propagation of Norad Element Sets. *Peterson AFB, CO, Spacetrack Report* (1980).

[54] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. 2015. Mobile edge computingâĂŤA key technology towards 5G. *ETSI white paper* 11, 11 (2015), 1–16.

[55] IP Access. 2023. Starlink vs. OneWeb: How Do They Compare? https://www.ipinternational.net/starlink-vs-oneweb-how-do-they-compare. (Accessed on October 7th, 2024).

[56] Isla McKetta. 2021. How Starlink's Satellite Internet Stacks Up Against HughesNet and Viasat around the Globe. https://www.speedtest.net/insights/blog/starlink-hughesnet-viasat-performance-q2-2021/. (Accessed on October 7th, 2024).

[57] Liz Izhikevich, Manda Tran, Katherine Izhikevich, Gautam Akiwate, and Zakir Durumeric. 2024. Democratizing LEO Satellite Network Measurement. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8, 1 (2024), 1–26.

[58] Jessica Watkins. 2024. Starlink Ground Station: Overview And Locations. https://starlinkspot.com/starlink-ground-station-locations.

[59] Ziye Jia, Min Sheng, Jiandong Li, Yan Zhu, Weigang Bai, and Zhu Han. 2020. Virtual Network Functions Orchestration in Software Defined LEO Small Satellite Networks. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.

[60] JON BRODKIN. 2022. Starlink is getting a lot slower as more people use it, speed tests shows. https://arstechnica.com/tech-policy/2022/09/ookla-starlinks-median-us-download-speed-fell-nearly-30mbps-in-q2-2022. (Accessed on October 7th, 2024).

[61] Josh Fomon. 2022. Here's How Fast Starlink Has Gotten Over the Past Year. https://www.ookla.com/articles/starlink-hughesnet-viasat-performance-q1-2022. (Accessed on October 7th, 2024).

[62] Josh Fomon. 2022. Starlink Slowed in Q2, Competitors Mounting Challenges. https://www.ookla.com/articles/starlink-hughesnet-viasat-performance-q2-2022. (Accessed on October 7th, 2024).

[63] Michael Kan. 2020. SpaceX's Satellite Internet Service Latency Comes in Under 20 Milliseconds. https://www.pcmag.com/news/spacexs-satellite-internet-service-latency-comes-in-under-20-milliseconds. (Accessed on October 7th, 2024).

[64] Mohamed M Kassem, Aravindh Raman, Diego Perino, and Nishanth Sastry. 2022. A browser-side view of starlink connectivity. In *Proceedings of the 22nd ACM Internet Measurement Conference*. 151–158.

[65] Simon Kassing, Debopam Bhattacherjee, André Baptista Águas, Jens Eirik Saethre, and Ankit Singla. 2020. Exploring the "Internet from Space" with Hypatia. In *Proceedings of the ACM Internet Measurement Conference*. 214–229.

[66] Farooq Khan. 2015. Mobile Internet from the Heavens. *arXiv preprint arXiv:1508.02383* (2015).

[67] Kirsty Needham. 2022. Musk's Starlink connects remote Tonga villages still cut off after tsunami. https://www.reuters.com/world/asia-pacific/musks-starlink-connects-remote-tonga-villages-still-cut-off-after-tsunami-2022-02-23/. (Accessed on October 7th, 2024).

[68] Zeqi Lai, Hewu Li, Yangtao Deng, Qian Wu, Jun Liu, Yuanjie Li, Jihao Li, Lixin Liu, Weisen Liu, and Jianping Wu. 2023. StarryNet: Empowering Researchers to Evaluate Futuristic Integrated Space and Terrestrial Networks. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1309–1324.

[69] Wiley J Larson and James Richard Wertz. 1992. *Space Mission Analysis and Design*. Technical Report. Torrance, CA (United States); Microcosm, Inc.

[70] Yuanjie Li, Hewu Li, Wei Liu, Lixin Liu, Yimei Chen, Jianping Wu, Qian Wu, Jun Liu, and Zeqi Lai. 2022. A case for stateless mobile core network functions in space. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 298–313.

[71] Lixin Liu, Yuanjie Li, Hewu Li, Jiabo Yang, Wei Liu, Jingyi Lan, Yufeng Wang, Jiarui Li, Jianping Wu, Qian Wu, et al. 2024. Democratizing {Direct-to-Cell} Low Earth Orbit Satellite Networks. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 791–808.

[72] Yingwei Luo, Binbin Zhang, Xiaolin Wang, Zhenlin Wang, Yifeng Sun, and Haogang Chen. 2008. Live and incremental whole-system migration of virtual machines using block-bitmap. In *2008 IEEE International Conference on Cluster Computing*. 99–106. https://doi.org/10.1109/CLUSTR.2008.4663760

[73] Alison Sider Micah Maidenberg. 2022. Delta Air Lines Tested SpaceX's Starlink Internet for Planes Delta CEO Says. https://www.wsj.com/articles/delta-air-lines-tested-spacexs-starlink-internet-for-planes-delta-ceo-says-11650316287. (Accessed on October 7th, 2024).

[74] Michael Baylor. 2018. With Block 5, SpaceX to Increase Launch Cadence and Lower Prices. https://www.nasaspaceflight.com/2018/05/block-5-spacex-increase-launch-cadence-lower-prices/. (Accessed on October 7th, 2024).

[75] Michael Kan. 2022. SpaceX's Starlink Now Serves Over 400,000 Subscribers. https://www.pcmag.com/news/spacexs-starlink-now-serves-over-400000-subscribers. (Accessed on October 7th, 2024).

[76] Michael Kan. 2022. Starlink Speeds Drop Significantly in the US Amid Congestion Woes. https://www.pcmag.com/news/starlink-speeds-drop-significantly-in-the-us-amid-congestion-woes. (Accessed on October 7th, 2024).

[77] Mitchell Clark, Richard Lawler. 2022. T-Mobile and SpaceX Starlink say your 5G phone will connect to satellites next year. https://www.theverge.com/2022/8/25/23320722/spacex-starlink-t-mobile-satellite-internet-mobile-messaging. (Accessed on October 7th, 2024).

[78] 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. In *Proceedings of the ACM on Web Conference 2024*. 2723–2734.

[79] Monica Alleven. 2021. Verizon taps Amazon's Kuiper for backhaul, rural connectivity. https://www.fiercewireless.com/wireless/verizon-taps-amazon-s-kuiper-for-backhaul-rural-connectivity. (Accessed on October 7th, 2024).

[80] Sebastian Moss. 2022. Thales Alenia Space wins EU feasibility study for 'Ascend' space data centers. https://www.datacenterdynamics.com/en/news/thales-alenia-space-wins-eu-feasibility-study-for-ascend-space-data-centers/. (Accessed on October 7th, 2024).

[81] United States Department of the Interior Minerals Management Service. 2023. Platform Locations in the Gulf. https://www.data.bsee.gov/Platform/Files/3060.pdf. (Accessed on October 7th, 2024).

[82] Oscar Ohlsson, Pontus Wallentin, and Claes-Göran Persson. 2020. Reducing mobility interruption time in 5G networks. *Ericsson, Stockholm Sweden* (2020).

[83] Oliver Peckham. 2021. Spaceborne Computer-2 Makes HPE's Case for Edge Processing. https://www.hpcwire.com/2021/09/02/spaceborne-computer-2-makes-hpes-case-for-edge-processing/. (Accessed on October 7th, 2024).

[84] Tobias Pfandzelter and David Bermbach. 2021. Edge (of the Earth) Replication: Optimizing Content Delivery in Large LEO Satellite Communication Networks. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 565–575.

[85] Tobias Pfandzelter and David Bermbach. 2022. Celestial: Virtual software System Testbeds for the LEO Edge. In *Proceedings of the 23rd conference on 23rd ACM/IFIP International Middleware Conference*. 69–81.

[86] Tobias Pfandzelter and David Bermbach. 2023. Can Orbital Servers Provide Mars-Wide Edge Computing? *arXiv preprint arXiv:2306.09756* (2023).

[87] Tobias Pfandzelter, Jonathan Hasenburg, and David Bermbach. 2021. Towards a Computing Platform for the LEO Edge. In *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*. 43–48.

[88] Planet Labs. 2023. Planet Imagery Product Specifications. https://assets.planet.com/docs/Planet$_{C}$ombined$_{I}$magery$_{P}$roduct$_{S}$pecs$_{l}$etter$_{s}$creen.pdf. (Accessed on October 7th, 2024).

[89] Maksym Planeta, Jan Bierbaum, Leo Sahaya Daphne Antony, Torsten Hoefler, and Hermann Härtig. 2021. MigrOS: Transparent Live-Migration Support for Containerised RDMA Applications. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 47–63. https://www.usenix.org/conference/atc21/presentation/planeta

[90] Reuters. 2022. Elon Musk to provide Florida with Starlink satellites in response to Hurricane Ian. https://www.reuters.com/world/us/elon-musk-provide-florida-with-starlink-satellites-response-hurricane-ian-2022-10-01. (Accessed on October 7th, 2024).

[91] Adam Ruprecht, Danny Jones, Dmitry Shiraev, Greg Harmon, Maya Spivak, Michael Krebs, Miche Baker-Harvey, and Tyler Sanderson. 2018. VM Live Migration At Scale. *SIGPLAN Not.* 53, 3 (mar 2018), 45–56. https://doi.org/10.1145/3296975.3186415

[92] Mahadev Satyanarayanan. 2017. The emergence of edge computing. *Computer* 50, 1 (2017), 30–39.

[93] Jinhyun So, Kevin Hsieh, Behnaz Arzani, Shadi Noghabi, Salman Avestimehr, and Ranveer Chandra. 2022. Fedspace: An efficient federated learning framework at satellites and ground stations. *arXiv preprint arXiv:2202.01267* (2022).

[94] Starlink. 2016. Application for Approval for Orbital Deployment and Operating Authority for the SpaceX NGSO Satellite System. https://cdn.arstechnica.net/wp-content/uploads/2016/11/spacex-Legal-Narrative.pdf. (Accessed on October 7th, 2024).

[95] Hammas Bin Tanveer, Mike Puchol, Rachee Singh, Antonio Bianchi, and Rishab Nithyanand. 2023. Making sense of constellations: Methodologies for understanding starlink's scheduling algorithms. In *Companion of the 19th International Conference on Emerging Networking EXperiments and Technologies*. 37–43.

[96] 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. (2024).

[97] Wojciech Tyczynski. 2015. Kubernetes Performance Measurements and Roadmap. https://kubernetes.io/blog/2015/09/kubernetes-performance-measurements-and/. (Accessed on October 7th, 2024).

[98] Wojciech Tyczynski. 2016. 1000 Nodes and Beyond: Updates to Kubernetes Performance and Scalability In 1.2. https://kubernetes.io/blog/2016/03/1000-nodes-and-beyond-updates-to-kubernetes-performance-and-scalability-in-12/. (Accessed on October 7th, 2024).

[99] Vallado, David A and Crawford, Paul and Hujsak, Richard and Kelso, TS. 2006. Revisiting spacetrack report# 3: rev 1. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*.

[100] Ranjan Sarpangala Venkatesh, Till Smejkal, Dejan S. Milojicic, and Ada Gavrilovska. 2019. Fast In-Memory CRIU for Docker Containers. In *Proceedings of the International Symposium on Memory Systems* (Washington, District of Columbia, USA) *(MEMSYS '19)*. Association for Computing Machinery, New York, NY, USA, 53–65. https://doi.org/10.1145/3357526.3357542

[101] Vivek Wadhwa and Alex Salkever. 2022. How Elon Musk's Starlink Got Battle-Tested in Ukraine. https://foreignpolicy.com/2022/05/04/starlink-ukraine-elon-musk-satellite-internet-broadband-drones. (Accessed on October 7th, 2024).

[102] Matthew Wiseman and Alice Bradley. 2019. Impact of the length of the sea ice-free summer season on Alaskan Arctic coastal erosion rates. In *AGU Fall Meeting Abstracts*, Vol. 2019. C13D–1345.

[103] Andrew Wooden. 2022. Lynk launches 'first commercial-ready cell-tower-in-space'. https://telecoms.com/514594/lynk-launches-first-commercial-ready-cell-tower-in-space-. (Accessed on October 7th, 2024).

A CALCULATING SIZE OF PROJECTED REGION

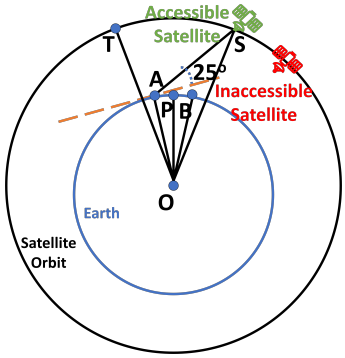


Figure 14: Space projected region calculation.

We use Figure 14 to calculate the size of the space projected region for given a LEO zone. In this figure, the outer circle represents the satellite's orbit, the inner circle represents the Earth with point O as the center. The arc AB is the intended LEO zone with the arc ST is its space projected region. Let the Earth radius be R, the satellite altitude be a, and the radius of the LEO zone be r. Point S represents the furthest a satellite can be from the LEO zone so that the entire LEO zone is accessible from the satellite (using the maximum elevation angle of 25^o). This gives us $OP = OA = OB = R$, $OS = R + a$. Further, $\angle SAO = 90^o + 25^o$. We observe that to calculate the size of the space projected region, we first need to calculate $\angle SOP$.

- In $\triangle$, we know the lengths of sides OA and OS, and $\angle SAO$. We use this to calculate $\angle ASO$ using the sine rule.
- Now with $\angle SAO$ and $\angle ASO$ are known, we calculate $\angle AOS$.
- We calculate $\angle AOP = \frac{AP}{R} = \frac{r}{R}$.
- Finally, we get $\angle SOP = \angle SOA - \angle AOP$.

With $\angle SOP$, we calculate the size of the space projected region, as well as its projection on Earth.

B EXAMPLES OF DIFFERENT LEO ZONES SHAPES

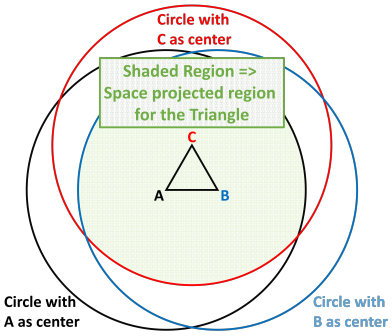


Figure 15: Space projected region for a triangular LEO zone.

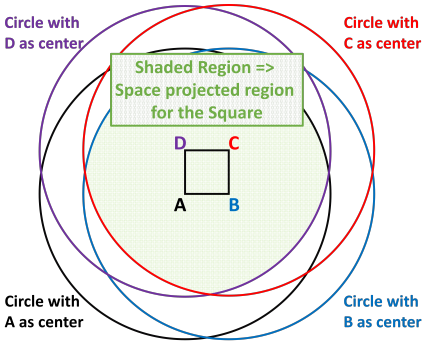


Figure 16: Space projected region for a square LEO zone.

We show what the space projected region will look like for non-circular shapes. To calculate the space projected region for any general shape LEO zone, the trivial approach would be to project the circular space projected region of every point in the zone and take their intersection. Recall that the space projected region for a point is a cone, thus when it is projected on the Earth, it would be circular. For polygonal shapes, this can be optimized by calculating the intersection of the space projected regions from the vertices. We show two examples - a triangle (Figure 15) and a square (Figure 16). The key takeaway here is that finding the space projected region for different shapes is feasible, even if more computationally involved compared to circular LEO zones. Krios preforms these computations when scheduling applications as well as to determine the triggers for handovers.