CELESTIAL: Virtual Software System Testbeds for the LEO Edge

Tobias Pfandzelter  
TU Berlin & ECDF  
Mobile Cloud Computing Research Group  
Berlin, Germany  
tp@mcc.tu-berlin.de

David Bermbach  
TU Berlin & ECDF  
Mobile Cloud Computing Research Group  
Berlin, Germany  
db@mcc.tu-berlin.de

ABSTRACT
As private space companies such as SpaceX and Telesat are building large LEO satellite constellations to provide global broadband Internet access, researchers have proposed to embed compute services within satellite constellations to provide computing services on the LEO edge. While the LEO edge is merely theoretical at the moment, providers are expected to rapidly develop their satellite technologies to keep the upper hand in the new space race.

In this paper, we answer the question of how researchers can explore the possibilities of LEO edge computing and evaluate arbitrary software systems in an accurate runtime environment and with cost-efficient scalability. To that end, we present CELESTIAL, a virtual testbed for the LEO edge based on microVMs. CELESTIAL can efficiently emulate individual satellites and their movement as well as ground station servers with realistic network conditions and in an application-agnostic manner, which we show empirically. Additionally, we explore opportunities and implications of deploying a real-time remote sensing application on LEO edge infrastructure in a case study on CELESTIAL.

CCS CONCEPTS
• Computer systems organization → Heterogeneous (hybrid) systems; Distributed architectures; • Networks → In-network processing.

KEYWORDS
satellite networks, LEO edge, software testbeds

ACM Reference Format:
Tobias Pfandzelter and David Bermbach. 2022. CELESTIAL: Virtual Software System Testbeds for the LEO Edge. In 23rd International Middleware Conference (Middleware ’22), November 7–11, 2022, Quebec, QC, Canada. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3528535.3531517

1 INTRODUCTION
Private aerospace and Internet companies, such as SpaceX\textsuperscript{1}, OneWeb\textsuperscript{2}, and Telesat\textsuperscript{3}, are launching tens of thousands of satellites to provide global broadband Internet access. Thanks to their low-Earth orbit (LEO) and free-space laser links, consumers can expect low-latency, high-bandwidth Internet access anywhere on Earth. New LEO satellite networks challenge not only the old satellite-based Internet access but terrestrial fiber as well [7, 53, 55].

Fueled by that development, it has also been proposed to implement computing resources within these LEO constellations, e.g., [8, 52, 65], to facilitate LEO edge computing, as is common with terrestrial multi-access edge computing (MEC) [22]. Edge resources located on communication satellites, which act as radio uplinks for LEO Internet subscribers, could provide low-latency application access from which especially rural areas and clients without a nearby cloud data center would benefit [8, 10, 49, 52].

LEO edge computing introduces novel challenges in application management: The number of potential satellite servers, with, e.g., the proposed Starlink constellation comprising more than 40,000 satellites at completion [28, 29, 40], will require new approaches to server management. Further, LEO satellites move at speeds in excess of 27,000km/h and ground equipment frequently needs to reconnect to new satellites, resulting in an ever-changing network topology. Finally, LEO edge software is subject to constrained compute resources and the harsh environment of space.

To solve these challenges, researchers will need to develop new middleware systems for application state management, request routing, and service offloading. As LEO edge computing currently exists only as a concept and any actual infrastructure is years away from implementation, emulated testbeds that go beyond the capabilities of simulation are needed to test and benchmark such systems. Emulating LEO edge infrastructure in the cloud, however, is non-trivial, as the number of satellite servers raises scalability and cost concerns, and the highly dynamic satellite network and environmental effects must be accurately reflected in the testbed.

In this paper, we thus raise the question of how we can evaluate arbitrary software systems for the LEO edge as accurately as possible and in a cost-efficient manner. To answer this question, we make the following contributions:

• We present LEO edge computing and the challenges of building and evaluating LEO edge software without access to actual infrastructure (§2).
• We introduce CELESTIAL, a novel LEO edge emulation tool based on microVMs and discuss how it addresses these challenges (§3).
• We evaluate these claims by deploying a latency-sensitive edge application on CELESTIAL (§4).
• In a case study on CELESTIAL, we evaluate different deployments of a distributed real-time data analysis service in the context of remote sensor networks to assess the opportunities and implications of the LEO edge environment (§5).

\textsuperscript{1}https://www.starlink.com/  
\textsuperscript{2}https://www.oneweb.world/  
\textsuperscript{3}https://www.telesat.com/
• We discuss threats to validity for our work and derive avenues for future work on Celestial and the LEO edge (§6).

We make our implementation of Celestial available as open-source to help future researchers validate their own applications and platforms. Our hope is that this will make the field of LEO edge computing more accessible and provide a starting point for systems research in this area.

2 BACKGROUND & RELATED WORK

In this section, we give an overview of the state of the art in large LEO satellite communication networks and describe the opportunities and challenges of the novel LEO edge computing paradigm. Furthermore, we discuss what it takes to build and test LEO edge software systems without access to LEO edge infrastructure, and where current simulators and testbeds fall short.

2.1 Large LEO Satellite Communication Networks

Satellite Internet access using geostationary orbits at altitudes in excess of 35,000km have been in operation for decades. Yet, their high communication delays and low bandwidth make them infeasible for most applications [14]. Advances in radio and laser technology [41], and progressively lower satellite launch costs in the last few years [39], have now enabled private companies such as SpaceX, Telesat, OneWeb, and Amazon to deploy high-bandwidth, low-latency satellite communication networks using thousands of satellites at altitudes between 500 and 1,500km, called the low-Earth orbit (LEO). Rather than only relaying radio signals from ground stations, these new satellites can use inter-satellite laser links (ISL) for communication between adjacent satellites. Given the vacuum in space, these ISLs can benefit from a ∼47% faster light propagation than in fiber cables [9, 28]. Point-to-point communication between two ground stations over the satellite network can thus incur less communication delay than with terrestrial fiber connections. As installing user equipment is also cheaper than installing fiber optic cables, the new large LEO satellite communication networks are expected to challenge not only traditional satellite Internet access but also terrestrial fiber [7, 9].

The core of a LEO satellite communication network is the actual satellite constellation. Such a constellation comprises shells of satellites, each shell at a different altitude and with different orbital parameters. Each of these shells consists of a number of orbital planes, evenly spaced around the equator. Within each plane are satellites that follow the same orbit, evenly spaced around that plane [50, 62].

Figure 1 shows an overview of the satellites in the planned phase I Starlink constellation, which comprises five such shells. The first shell has 1,584 satellites at altitudes of 550km and is split into 72 planes of 22 satellites each. Each of these orbital planes is inclined at an angle of 53° to the Earth’s equatorial plane. Additional shells at higher altitudes provide greater areas of coverage at the expense of additional network delay [28, 40]. The ISLs of such a constellation are likely to be arranged in a ≈GRID pattern, where each satellite keeps a link to its predecessor and successor within its plane as well as to one neighbor each in the two closest adjacent planes. This allows any two ground stations to connect directly to each other over the satellite network [9].

Figure 1: Overview of the planned phase I Starlink constellation with five shells of 1,584 satellites at 550km (turquoise), 1,600 at 1110km (orange), 400 at 1130km (blue), 375 at 1275km (pink), and 450 at 1325km (green) altitude. Colored lines illustrate ISLs, bright green lines show possible ground-to-satellite links for a ground station [40].

2.2 Bringing Compute to the LEO Edge

The edge computing paradigm is the answer to a growing demand and need to process data close to its origin rather than in distant cloud data centers [6, 64]. In fog computing [11] and the terrestrial MEC architecture [22], compute resources are embedded within the access network, i.e., close to clients, the network edge. Here, edge applications run on virtualization infrastructure, which makes them available with high bandwidth and low latency, with lower network costs, and with decreased privacy and security risks [47, 56].

In large LEO satellite communication networks, the network edge is the satellite constellation itself – the last provider-operated hop before packets reach user equipment. Uplinks for groups of ground stations converge in a satellite, making it an efficient location for shared computing resources close to clients [49]. Possible applications for LEO edge computing include CDN replication to reduce bandwidth usage in the network and access latency for clients, meetup servers for multi-user interaction such as low-latency gaming or video conferencing, or processing space-native data [8, 10, 49] – a detailed overview of the architecture of such an application is given in our case-study in §5. Given the current skew of cloud and cloudlet data centers being placed close to metropolitan areas, LEO edge computing could help meet the needs of rural areas through the global coverage of LEO satellite networks.
A number of challenges still lie ahead on the path to global LEO edge computing: First, satellite constellations themselves are still being built out and do not yet provide uninterrupted global coverage. As a result, there are no system traces from real world LEO satellite constellations. Second, none of the communication satellites that are already in operation at the moment provide sufficient additional compute resources that could be sold to customers, as satellites are limited by payload restrictions and launch costs. Although multi-tenant satellites, e.g., the推迟 fractionated satellite [13] or the Tiansuan satellites [63], have been researched, to the best of our knowledge, no operational compute platform for LEO satellite networks exists at the moment. Third, the deployment and operation of such resources in LEO introduce additional engineering challenges in power consumption, waste-heat management, or radiation hardening [8, 46, 61]. Fourth, how to abstract from the highly mobile and unique infrastructure using LEO edge computing platforms is still actively being researched, e.g., [10, 52].

2.3 Open Challenges in Building and Testing LEO Edge Software Systems on Earth

A key part of systems research for the LEO edge is the evaluation of software systems in a realistic environment, e.g., through functional testing and benchmarking. While algorithms can be evaluated in simulation, studying the behavior of concrete software systems requires a realistic runtime environment. We believe that enabling researchers and practitioners to quickly and affordably run virtual LEO edge computing infrastructure in the cloud will accelerate both systems research and the actual development and deployment of infrastructure as operators gauge industry and research interest. Yet, building such testbeds for large-scale, highly dynamic infrastructure requires addressing three specific research challenges: How can we ensure that a testbed offers an accurate representation of real LEO constellations? Which abstractions are necessary in the design of a testbed so that arbitrary software systems can be deployed on it? Finally, how can we achieve this with cost-efficiency at scale?

Accurate Emulation of LEO Constellations. A fundamental difference of the LEO edge compared to terrestrial edge or cloud computing is the high mobility of satellites in relation to Earth that result from their low orbits. A complete LEO edge infrastructure is constantly evolving, with network connections and characteristics between servers changing by the second. Furthermore, ground stations also frequently switch their uplink to their closest satellite as a result of this mobility. All applications and platforms developed for the LEO edge need to consider this and proactively replicate data and services. These characteristics of the network influence the performance of edge applications, which are often latency or bandwidth sensitive, and must thus also be part of a LEO edge computing testbed. This should be based on detailed models of Earth and space so that hand-off and data migration techniques can be evaluated as accurately as possible.

Satellite servers will likely use commercial, off-the-shelf compute hardware, e.g., as HPE does with their Spaceborne Computers onboard the International Space Station [21, 58]. Although the effects of the Van Allen radiation belts are negligible in lower orbits, servers will be subject to some single event upsets caused by intermittent galactic cosmic rays that can normally be absorbed by Earth’s atmosphere and magnetic field [43]. HPE has shown that this can be remediated with standard software and hardware mechanisms, yet at the cost of temporary performance degradation or full shutdows. Such failure will impact any software running on a satellite and developers will want to test their applications against these scenarios. Especially service orchestrators on top of LEO edge infrastructure need to adapt and react quickly.

Support for Arbitrary Software Systems. A LEO edge computing testbed should enable development and evaluation of any kind of LEO edge software and should not be restricted to certain programming languages, frameworks, or deployment models. The emulator should thus accurately reflect blank-slate servers, especially as we still face unknowns regarding middleware platforms and applications.

When considering the development of platforms, great care should be taken to ensure that a virtual testbed does not impose restrictions on the kind of technologies that can be evaluated within, e.g., a container-based approach where each satellite server is emulated with a single container would impose limitations on testing container-based platforms.

Cost-Efficient Scalability. Large LEO satellite constellations can comprise tens of thousands of satellites and serve millions of clients. Building and scaling applications and platforms for such infrastructure is a difficult challenge and a LEO edge testbed should be able to serve as a way to perform scalability evaluation. The testbed must thus also scale for large constellations and provide the means to emulate such infrastructure.

However, users should not be expected to also provide tens of thousands of physical computers for such an evaluation. Rather, the testbed should be cost-efficient in a way where more than one satellite server can be emulated on one host. Further, it should also let the user evaluate only a part of the complete constellation, e.g., to support rapid prototyping and testing of a constellation subset over a certain geographical area or to save experiment costs.

2.4 Related Work

In the edge computing, where applications are widely distributed and physical infrastructure is often inaccessible, researchers commonly rely on virtual testbeds, network simulators, and edge simulators for testing.

Testbed tools such as Fogbed [16], EmuFog [44], or MockFog [30, 31] allow users to create and manipulate virtual infrastructure that mimics a distributed, heterogeneous edge-cloud continuum. Fogbed and EmuFog, however, do not support an emulation of highly dynamic network topologies as required for large LEO satellite constellations. Furthermore, applications are deployed as Docker containers, which makes it unsuitable for evaluating novel platforms as it limits supported software. MockFog supports dynamic network changes and uses a dedicated cloud virtual machine for each compute node, which imposes fewer restrictions on the kinds of software it supports, yet such flexibility comes at a price: With a dedicated cloud virtual machine for each satellite server, we cannot achieve a cost-efficient emulation for large LEO constellations. Similar concerns can also be raised for further IoT and edge computing testbed tooling [4, 5, 20, 34, 35].
3.1 Accurate Emulation of LEO Constellations

At the heart of Celestial, the Constellation Calculation component updates the state of the satellite network periodically, including the positions of satellites and ground stations, network link distance and delays, and shortest paths between nodes. This is based in large parts on the SILLEO-SCNS network simulator [42] for large LEO satellite constellations, which we extend with support for the SGP4 simplified perturbations models. Additionally, we use more efficient implementations of Dijkstra’s algorithm [18] and the Floyd-Warshall algorithm [23] to calculate the shortest network paths within the constellation and their end-to-end latency.

SGP4 is the state-of-the-art in calculating the positions of satellites and takes perturbations caused by atmospheric drag, the Earth’s shape, and gravitational effects from Moon or Sun into account [36]. The model input parameters can be obtained from the database of NORAD two-line element sets (TLE)\(^5\) for satellites already in orbit or calculated based on simple parameters such as a satellite shell’s inclination and altitude. To limit side effects and ensure repeatable testing, all parameters are passed within a single configuration file. This includes network parameters, such as ISL bandwidth, compute parameters that describe the allocated resources for satellite and ground station servers, orbital parameters for satellite shells to support different kinds of constellations, and ground station locations.

Celestial then uses satellite and ground station positions to calculate ground station uplinks and the satellite network topology. Here, ISL connectivity depends on the line of sight between two adjacent satellites, e.g., if a possible laser link drops below a certain altitude, the Earth’s atmosphere may refract that laser, causing an intermittent loss of connectivity. Similarly, ground stations can only communicate with satellites that are above a configurable minimum elevation above the horizon. In our tests, these calculations could be completed within one second even on a standard laptop. The results are then transferred to the hosts, where network delays and bandwidth constraints between the satellite servers are emulated using tc\(^6\) and tc-netem\(^7\). Emulated network delays are injected with a 0.1ms accuracy and any latency between hosts is taken into account, yet this only works if this latency is low enough, e.g., hosts are located in the same datacenter. Additionally, emulated servers can still reach the Internet through the host, e.g., to store experiment data in some central location.

We isolate microVMs in dedicated cgroups to gain more finely grained control over the CPU cycles a server process is allowed to use, making the emulation of severely constrained satellite servers possible. Through an API, users can change machine parameters at runtime and even terminate and reboot machines to model faults, e.g., caused by radiation.

Celestial also provides an optional animation component that visualizes the state of the constellation during the emulation run, e.g., Fig. 1 was generated by this component. We believe that this can be a great help in understanding the characteristics of satellite mobility and networking as well as the effects on their software systems, especially for developers who are new to satellite networks.

---

\(^5\)https://celestrak.com/NORAD/elements/

\(^6\)https://man7.org/linux/man-pages/man8/tc.8.html

\(^7\)https://man7.org/linux/man-pages/man8/tc-netem.8.html
In Celestial, we take all effects into account that are currently conjectured to have a significant impact on software systems running on the LEO edge, namely dynamic network delays, bandwidth restrictions, constrained compute and storage resources, and service interruption through environmental effects such as radiation [8, 52]. When additional effects are studied and traces from real LEO edge deployments become available, researchers will be able to quickly incorporate the necessary changes in Celestial: First, the core Constellation Calculation component that dictates the effects that are emulated in the rest of the system comprises only 971 lines of Python source code that can be easily customized and extended. The separation from Machine Managers allows the resulting networking and machine parameters to be sent to host machines without modification. Second, tc-netem offers advanced network emulation features that are not currently used in Celestial, such as packet loss or duplication, delay distributions, packet corruption, or packet reordering. If such characteristics will be useful for LEO edge testbeds in the future, they can be added with only small changes to the Celestial codebase. Third, we forego the use of complex dependencies such as orchestrators (e.g., Kubernetes or Mesos) and overlay network tooling (e.g., Flannel or Calico) to reduce the complexity of Celestial. While we rely on some of their underlying technologies, such as the firecracker-containerd plugin, not including such complex software simplifies prototyping new features and reduces maintenance efforts of our implementation, especially considering the required changes to such dependencies.

3.2 Support for Arbitrary Software Systems
All satellite and ground station servers are emulated with Firecracker [1] microVMs that run on the Celestial hosts, managed by the Machine Manager components. Firecracker microVMs provide a sub-second boot time and support for microVM suspensions, and can use configurable Linux kernels and root filesystems. Celestial thus gives users the control over kernel features, installed software, and programming and deployment models for their applications. The process of compiling filesystems, starting (cloud) hosts, and uploading the necessary files can be automated using common orchestration tools such as Ansible\(^1\), yet it is highly user-specific, so we do not include it in Celestial. Crucially, satellite servers are thus provided as a blank-slate and users may even set up container technologies such as Docker within their emulated servers. This is of considerable importance for systems researchers, as it also facilitates the evaluation of container-based application orchestration using tools such as Kubernetes or FaaS platforms [10, 48].

To aid the development of applications and platforms on Celestial, it includes two additional components: First, each Celestial host provides a local DNS server that can resolve microVM network addresses with a custom DNS record. Applications can simply query the A records for, e.g., 878.0.celestial to get the network addresses of satellite 878 in the first shell. Applications thus do not have to be aware of the underlying IP address space calculation for Celestial’s virtual network interfaces.

Second, Celestial hosts run an HTTP server that provides information on satellite positions, network paths between satellites, constellation information, and more to the emulated satellite servers. This information is sourced from a central database on the Celestial coordinator that is updated by the Constellation Calculation. Application developers can leverage this API to quickly test their applications on different kinds of LEO constellations without having to implement a custom model of satellite movement and network behavior. In a real LEO edge computing scenario, we expect such information to be available from a central source provided by the satellite network operator or from public sources such as a TLE database. However, users are free to use only a subset or none of these APIs, e.g., when testing their own models.

3.3 Cost-Efficient Scalability
To emulate arbitrarily large LEO and complex LEO satellite constellations with thousands of satellite servers, Celestial supports horizontal scale-out across many hosts over which microVMs are distributed. For this, users may simply instantiate necessary cloud infrastructure that can be terminated once experiments and tests are completed. Celestial automatically creates an overlay network using WireGuard [19], thus connecting hosts and offering routing between microVMs. Further, the use of microVMs allows for high-over-provisioning as well as collocation of many emulated satellite servers on few physical servers. In Celestial, we use the fact that all satellite servers are identical to our advantage and de-duplicate microVM root filesystems using a common immutable disk image in addition to an overlay for each microVM, allowing us to save on storage space and improve performance.

Celestial can emulate large and complex LEO satellite constellations with thousands of satellite servers, yet not every test or evaluation requires emulating each individual satellite server at the same time: As an optional feature, we introduce a configurable bounding box, a geographical area on Earth to which emulated satellite servers are limited. As satellites are mobile, they can quickly move in and out of this bounding box and may only stay relevant for evaluation for a few minutes. To free up resources, satellite microVMs are suspended when they move out of the bounding box and re-activated when they come back into it. The underlying idea is that in edge computing, clients will want to use edge servers that are in their proximity, and distant edge servers will thus not need emulation. To give an example, in §4, we use the bounding box to only emulate satellites over North Africa, where our clients are located, to save resources. Celestial also helps the user configure their bounding box in a manner that makes sure that available resources meet the demand from the emulation based on per-microVM resources and bounding box area. Note that this bounding box does not affect network path calculation, as the shortest network path between two ground stations may not follow the line of sight. This allows algorithms, applications, and platforms for large LEO satellite constellations to be tested cost-efficiently on a minimal subset of servers before moving to an emulation of the entire constellation.

\(^1\)https://github.com/firecracker-microvm/firecracker-containerd
\(^2\)https://www.ansible.com/
4 DEPLOYING AN EDGE APPLICATION ON CELESTIAL

To evaluate CELESTIAL’s performance, accuracy, and scalability, we first deploy an example LEO edge application, namely a multi-user interaction between users in West Africa as presented by Bhatacherjee et al. [8]. In this example, which we illustrate in Fig. 3, three users located in Accra, Ghana; Abuja, Nigeria; and Yaoundé, Cameroon require a common meetup-server for their application. While their nearest available cloud data center is located in Johannesburg, South Africa. Dashed lines indicate the additional hops needed to reach the cloud datacenter [8].

![Figure 3: Three clients in Accra, Ghana; Abuja, Nigeria; and Yaoundé, Cameroon, require a common meetup-server for an interactive application. This meetup server may be located in a satellite server or the nearest cloud datacenter in Johannesburg, South Africa. Dashed lines indicate the additional hops needed to reach the cloud datacenter [8].](image)

4.1 Experiment Setup

We assume both laser propagation in a vacuum and propagation of ground-to-satellite RF links at speed of light $c$ [28, 59] and a 10Gbps bandwidth connection for ISLs and radio links. We allocate four CPU cores and 4GB memory for each client and the tracking service, while each satellite server as well as the cloud video bridge are allocated two CPU cores and 512MB memory. To limit resource usage and the number of satellite servers the tracking service must consider, we draw a bounding box as shown in Fig. 3. To minimize the impact of clock drift in delay measurements between our clients, we schedule them to run on the same CELESTIAL host and instruct them to use a shared PTP clock. In a preliminary baseline evaluation, we find that our clients and bridge server incur a 1.37ms median processing delay (3.86ms standard deviation) that we must take into account when comparing measured end-to-end latency and expected network distance. We expect our client measurement software, packet duplication, packet forwarding, and clock drift to cause this jitter.

Our experiments are conducted on three Google Cloud Platform N2-highcpu instances with 32 cores and 32GB memory each, placed in the `europe-west3-c` zone, connected in a private virtual network, and using an installation of Ubuntu 18.04. We measure a network delay of 0.2ms between those machines, yet this delay is already taken into account by CELESTIAL when emulating microVM network distance. Although larger instances are available that may allow us to host our testbed on only a single cloud server, our goal is to show the horizontal scalability of CELESTIAL. While CELESTIAL estimates 137 required CPU cores given satellite density and bounding box size, we use only 96 CPU cores to test its over-provisioning capabilities. As our application actually uses resources on only one satellite server at a time, we expect only a small load on our CELESTIAL servers. In addition, we run our CELESTIAL Coordinator on a GCP C2 instance with 16 cores and 64GB memory and choose an update interval of two seconds. Each experiment is ten minutes long and repeated three times to validate its reproducibility.

4.2 Results

We start by giving an overview of the results of our experiments by presenting the results of a randomly selected run. We can then validate the accuracy of the network conditions in our CELESTIAL testbed by comparing it to the simulated network, analyze the reproducibility of our experiments on CELESTIAL, and finally check resource efficiency.\(^\text{10}\)

We show the cumulative distributions of measured end-to-end latency between client pairs in Fig. 4. For at least 80% of the duration of the video conference, end-to-end latency is below this maximum RTT of 16ms for the satellite servers and 46ms for the cloud server, confirming our expectation that satellite servers can provide a considerable QoS improvement for clients in latency-sensitive applications.

While available bandwidth is an unlikely factor for increased latency in the remaining 20% given the small size of our video stream, processing delay may add several milliseconds. Additionally, we cannot assume an optimal server to be selected at all times as, in some instances, the satellite constellation may change noticeably.

\(^{10}\)We include additional results in a tech report on CELESTIAL [51].
during the five-second update interval used in our application, e.g., with a particular uplink satellite becoming unavailable.

We further observe that only satellites in the two shells with the lowest altitude and highest density are ever selected as satellite servers, as they are more likely to have an optimal connection to all three clients at the same time. In deploying LEO edge infrastructure, it might thus make sense to start with the densest and lowest altitude shell.

Accuracy. To investigate whether the measured latency over time is accurate, we compare expected network latency as calculated by our tracking server, which includes the 1.37ms median processing delay. As an example, we show this comparison for the path from Abuja to Accra using the cloud server over the course of one evaluation in Fig. 5.

Both curves follow the same general trend and changes in calculated network latency are reflected in measured end-to-end latency. Again, we note that the tracking server only chooses a new server at the coarse interval of five seconds and intermittent changes in infrastructure that can cause spikes in end-to-end latency are thus not reproduced in expected network distance. Additionally, the jitter in our measurements is likely caused by processing as we observe it in our baseline measurements as well. We thus conclude that simulated network distances are accurately reflected in the emulated testbed.

Reproducibility. Furthermore, we can investigate whether our results are reproducible by looking at the results from the three repetitions we carry out for each experiment. As an example, we plot the measured end-to-end latency from Yaoundé to Abuja using the cloud server across the three repetitions in Fig. 6.

We can observe that results for all three runs follow the same trends and even the spike in measured latency after the first minute of the experiment can be reproduced. Since users can provide an arbitrary but firm starting point for their testbed emulation, CELESTIAL offers a repeatable environment that enables reproducible tests and benchmarks.

Efficiency. Finally, in a separate test we trace the CPU and memory usage on our CELESTIAL hosts for a glimpse into the resource efficiency of our testbed. Specifically, we look at the host under the highest load, which is that on which all our clients run for accurate time synchronization, in addition to a third of all satellite servers. We show its CPU and memory utilization in Figs. 7 and 8, respectively.

At the beginning, we see a spike in CPU usage caused by CELESTIAL’s Machine Manager as it starts setting up the host and network environment, followed by an even larger spike as Firecracker microVMs are starting to boot up at the beginning of the emulation run. CPU usage then decreases to below 5% as the clients prepare for the experiment, e.g., by synchronizing clocks and reading the workload traces. Then, during the experiment, we see total CPU usage from microVMs on the order of 10%. Considering that three clients and one tracking server with four allocated CPU cores each, and one active bridge server with two allocated cores run a demanding workload, and at least 25 additional satellite servers idle, we can conclude that CELESTIAL is efficient on CPU resources.
as it can benefit from over-provisioning. Additionally, note that Celestial’s Machine Manager itself consumes few CPU resources after the initial start up, an average of 0.2% with a slightly higher load every two seconds as the constellation is updated.

On the other hand, Celestial’s Machine Manager uses up to 4.5% of the host’s available memory from the start of the simulation, although that number decreases after the demanding initial setup. Firecracker microVM memory usage increases linearly with the number of booted microVMs, regardless of whether they are suspended or not, as each keeps a virtio memory device that blocks a fixed portion of the host’s memory for the VM. As microVMs are only suspended when their corresponding satellites move out of the bounding box, their memory is not released. While this has not been an issue in any of our experiments, because microVM memory usage stays below 20% even on hosts with comparatively little available memory, Firecracker microVMs can also be configured to use ballooning to allow the host to reclaim unused memory from the VMs.

Finally, we also note the cost of our testbed: For our three hosts and one coordinator, a 10-minute experiment with an additional five minutes for setup and data collection yields a total cost of $3.30 on Google Cloud Platform. For comparison, creating 4,409 f1-micro virtual machine instances, with one for each satellite server, costs at least $539.66 for 15 minutes [26].

5 CASE STUDY: REAL-TIME OCEAN ENVIRONMENT ALERTS WITH REMOTE SENSORS

Using Celestial, we can empirically investigate the potential of the LEO edge for applications that may benefit from running at the network edge. One class of such applications are real-time monitoring services, e.g., in the context of industrial IoT or environment observation. The National Oceanic and Atmospheric Administration’s (NOAA) Deep-ocean Assessment and Reporting of Tsunamis (DART) project uses remote sensing on stationary buoys located in the Pacific to detect early tsunami warning signs [25, 45]. Given their remote locations, these sensors cannot communicate over terrestrial networks, but instead use the Iridium LEO satellite constellation. Sensor data is sent to the Pacific Tsunami Warning Center on Ford Island, Hawaii, where it is processed centrally. This system’s architecture is illustrated in Fig. 9.

We use a system inspired by DART to evaluate if and how LEO edge computing can assist real-time ocean environment monitoring for use-cases such as Tsunami warning. In our experiments, 100 data buoys in the Pacific Ocean transmit sensor readings over the Iridium satellite network. The Iridium satellite network has a single shell, 66 satellites in 6 planes at a 780km altitude, in a polar orbit (90° inclination) and spaced evenly only around half the globe (180° arc of ascending nodes) so that satellites descending their orbit cover the other half [17]. As a result of this spacing, the Iridium constellation cannot provide ISLs between the first and last orbital planes, which is reflected in our testbed. The readings, grouped by sensor location and type, are used to predict weather and environmental events with a long short-term memory (LSTM) neural network [37]. Results are distributed to ships and islands in the vicinity of the sensor, using a total of 200 locations. We show this topology in Fig. 10. We compare two different deployments of the inference service: First, we deploy it in a central ground station server at the location of the Pacific Tsunami Warning Center on Ford Island, Hawaii. Second, we deploy the inference service on each of the Iridium satellites, facilitating device-to-device communication.

5.1 Experiment Setup

All components of our experiments run on a Celestial testbed. Sensor data is sent at a one-second interval over UDP. Again, we use shared PTP clocks to minimize the impact of clock drift. Both
The effect of the lack of ISLs between first and last orbital plane of the Iridium constellation can be seen in both deployments: Over the course of the experiment, locations in the West Pacific region (Asia and Oceania) tend to connect to uplink satellites from the last orbital plane, while those in the Americas connect to the first. Consequently, any requests between those locations must be routed through satellites near the poles, increasing the communication delay. In the central processing deployment, this leads to a higher observed latency in this area over the course of the experiment, as all sensor data are routed in this manner. In higher latitudes, which are nearer to the North Pole, this effect is not as pronounced. With satellite servers, however, data is only routed across shorter geographical distances. Increased communication delays can thus only be observed when sensor station and data sink connect to opposite planes of the constellation, which happens less frequently when both are geographically close.

### 5.3 Opportunities and Implications for LEO Edge Applications

The results of our experiments show that LEO edge computing can improve QoS for remote-sensing applications where communication delay is the limiting factor. Without sending all data to a central processing facility, such as a cloud datacenter, and instead processing it on the communication path, significant latency improvements can be achieved. For this machine-to-machine communication use-case, on-device processing could not yield such an improvement: In case of processing on data buoys, sensor data would need to be sent back to buoys of the same group, incurring additional delays. Processing directly at the data consumers, i.e., island or ship ground stations, requires performing the same computation on several machines.

Please note that our experiment assumed rather small data volumes being sent: Since communication delay is the main cost factor in end-to-end latency, larger data volumes will increase the latency difference further. Also, larger data volumes will at some point make it inefficient to transmit all data for centralized processing – similar to bandwidth limits in terrestrial edge computing [47]. Similar to terrestrial edge computing, however, centralized processing may be more efficient for scenarios with low data volumes that can tolerate the higher end-to-end latency.

## 6 DISCUSSION & FUTURE WORK

In our evaluation of Celestial, we find that it fulfills our original requirements for a LEO edge testbed. In this section, we discuss threats to validity as well as limitations of our work and derive avenues for future work on Celestial and the LEO edge.

### 6.1 Limits to Scalability

In our experiments, we show that Celestial is horizontally scalable across multiple servers and that we can run an application in a realistic environment of large LEO edge infrastructure. We use both over-provisioning of our hosts and a bounding box to achieve this in a cost-efficient manner, but there may be use-cases where neither is an option. When emulation of the entire infrastructure is needed and all satellite and ground station servers run at their full capacity, host infrastructure must be sized appropriately, e.g., requiring 36TB.
of memory when emulating a full constellation of 4,400 satellites with 8GB memory each.

While Celestial is designed to scale out across as many hosts as necessary, we must assume that such scale comes with caveats. The network connection between hosts could, for example, become a bottleneck if inter-satellite communication bandwidth exceeds available physical bandwidth. Some of these effects could be mitigated in Celestial by dynamically migrating satellite server microVMs across hosts to optimize communication and resource provisioning, using a more advanced scheduler such as FirePlace [3].

6.2 Resource Isolation in microVM Collocation
Choosing microVMs over alternative technologies allows us to provide an application-agnostic runtime environment while also achieving cost-efficiency through collocation of multiple machines on one physical server. Nevertheless, that collocation can come at a price if resources are not appropriately available and microVMs start competing for resources. While satellite servers are independent of each other, with each server placed on an individual satellite, microVMs on one host may affect each other’s performance if their processes are scheduled on the same physical host CPU core. In practice, these effects cannot be easily circumvented without attaching microVM processes to host CPU cores strictly, which limits scalability as it inhibits over-provisioning of hosts. Yet, this can only become a larger problem once resources required by satellite servers exceed the host’s resources, which can be mitigated by scaling Celestial hosts vertically through additional CPU cores or memory, or horizontally by adding further host machines.

6.3 Impact of microVM Suspension
Celestial’s bounding box allows for a smaller testbed footprint by suspending microVMs of satellite servers that move outside a specified area. In our example application, we have seen how this reduces the load on the Celestial hosts without impacting the application. There may be some cases where microVM suspension has unwanted side effects on the user’s application. If the satellite server software is expected to change its state based on its location, e.g., a CDN service that needs to proactively replicate files before it is in reach of a ground station, it will not be able to do so while the machine is suspended and will potentially have to catch up to missed updates once it is activated again. To avert this, a user may increase the size of their bounding box, possibly to cover the entire earth so that no microVM is ever suspended. Note that this requires more host resources and hence increases cost.

6.4 Assumptions on Hardware Architecture
To work on commodity hardware, Celestial assumes satellite servers to have the same hardware architecture as common terrestrial servers, i.e., the x86-64 architecture supported by Firecracker. In reality, we cannot currently know the hardware which will be used on the LEO edge. Specific hardware features such as special instructions or real-time guarantees can thus not be emulated on Celestial at the moment and would require full hardware virtualization or at least emulation of a subset of instructions. Still, even if future LEO edge operators were to go the unlikely path of developing a unique compute infrastructure for satellite servers, Celestial could still provide a realistic environment to evaluate algorithms and programming models.

6.5 Emulating External Factors
Although Celestial can emulate satellite server degradation, e.g., caused by radiation, there are many other external factors that can impact satellite constellations. To give one example, satellites may not always follow completely deterministic orbits. Starlink’s
satellites are equipped with ion thrusters to adjust their orbits, which can be necessary to dodge space debris or other satellites [2, 24]. This has effects on the network as well, as physical distances change or ISLs can become unavailable during such maneuvers.

Adverse weather conditions can have an effect on the radio link between a ground station and satellites, with radio dishes overheating [12] and rain causing refraction of radio waves [54]. Ground station equipment may also be mobile, e.g., if installed on a plane or car, which must be taken into account when selecting uplink satellites. Such factors may impact LEO edge applications, and it can be helpful to test their effects ahead of application deployment.

The separation of Constellation Calculation and Machine Managers in Celestial will allow researchers to incorporate new models quickly, as only the calculation component must be extended. The resulting networking and machine parameters can then be sent to the host machines without modification. Additionally, tc and \texttt{tc-netem} offer advanced network emulation features that can be used in Celestial in the future with only small changes to its codebase, such as packet loss or duplication, delay distributions, packet corruption, or packet reordering. Whether these emulated characteristics will be useful for LEO edge testbeds depends on future research on LEO satellite network measurements.

While no efficiently emulated testbed can ever be fully accurate with respect to the environment it emulates, we believe that the current feature set of Celestial accurately reflects what is known about the LEO edge today. As future research on LEO satellite networks will likely yield additional insights on the effects of external factors, Celestial can easily be adapted or extended in all possible directions thanks to its small and simple codebase.

### 6.6 Lack of Validation Data

With Celestial, one of our goals is to emulate the LEO edge accurately, yet the only method to verify this is to compare our measurements with simulation results. What the research community is still missing are realistic data traces for large LEO satellite constellations, as network operators guard their data to keep a competitive edge. The recent SatNetLab proposal for a research platform for global satellite-based networks [57] may change this in the future, and we hope to adapt Celestial’s network calculation as new data becomes available.

### 6.7 Future Work on LEO Edge Systems

With its scale and dynamic topology, the LEO edge poses significant challenges for applications. To simplify the deployment of services, systems researchers should address these challenges with application platforms that abstract from the underlying infrastructure. The number of satellite servers will require new coordination approaches to guarantee service consistency. Further, the servers’ limited resources must be allocated efficiently, while they are possibly even shared by multiple tenants [52].

The high mobility of LEO satellites introduces the additional challenge of state management: Clients will frequently need to connect to a new satellite edge server, and any server-side state must be migrated accordingly. Bhattacheryee et al. [8] have proposed the concept of “virtual stationarity”, where such state is migrated between satellites based on their locations relative to Earth, so that data appears to be in the same location from a client perspective. Further, such state management requires routing the clients’ requests to the correct satellite server, which is made more difficult by the dynamic network topology.

Celestial itself does not include any strategies for state management, request routing, or service management, as it is intended as a testbed on which future systems implementing such strategies can be evaluated.

### 6.8 Feasibility of the LEO Edge

Finally, Celestial’s utility depends heavily on the future of the LEO edge. While considerable resources are committed to the development of large LEO satellite networks to provide global Internet coverage, LEO edge computing has so far only received limited attention from research and industry. If current trends in this field do not come to fruition and satellite network operators do not see sufficient financial incentives in operating LEO compute infrastructure, Celestial can only be used for theoretical evaluation.

Nevertheless, we also see this as an opportunity to discover those incentives: Celestial makes it possible to easily evaluate possible LEO edge applications and to bootstrap the development of LEO edge infrastructure.

### 7 CONCLUSION

In this paper, we have motivated the need for a testbed for the LEO edge that enables systems researchers, application developers, and platform designers to test and evaluate their LEO edge computing software on Earth. With Celestial, we have answered how such a testbed can be built for accuracy, in an application-agnostic manner, and with cost-efficiency and scalability in mind. In support of those claims, we have used Celestial to deploy a LEO edge application from the existing body of research. Additionally, we have shown in a case study how we use Celestial to empirically evaluate the potential of the LEO edge for remote sensing applications.

Finally, we have laid out interesting avenues for future work: With Celestial, researchers will now be able to build and test platforms and systems for the LEO edge that address the challenges of state management, resource allocation, or request routing. To further improve the accuracy of emulated testbeds, experiences and data of real LEO edge deployments are needed. We hope that Celestial, which we have published as open-source, can be of great use in this further research on and development of LEO edge computing.

### ACKNOWLEDGMENTS

We thank our anonymous reviewers, our anonymous shepherd, and our colleague Dr. Jonathan Hasenburg for their valuable feedback on this work. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 415899119. This work is supported by the Google Cloud Research Credits program (GCP202443755) and by the AWS Cloud Credit for Research program.
