GEO-AWARE SATELLITE ON-BOARD DATA INFRASTRUCTURE SERVICES

INVENTORS:

THOMAS SANDHOLM

SAYANDEV MUKHERJEE

BERNARDO HUBERMAN

BACKGROUND

Advances in both communication technology such as phased-array antennas and the economics of satellite launches have made Low-Earth Orbit (LEO) satellite services like Starlink, Kuiper, and OneWeb a promising approach to providing connectivity in remote or sparsely populated areas. It is also expected that the next evolution in LEO communication will move parts of the core network on board the satellites and leverage speed-of-light free-space optical links between satellites to reduce both end-to-end latency and the number of ground base stations.

One problem is that satellite network operators (SNOs) have neither infrastructure nor spectrum licenses in common with mobile network operators (MNOs), all of whose infrastructure and licenses so far have been terrestrial. Thus, the success of satellite-based connectivity depends on MNOs negotiating suitable licensing agreements with SNOs. The history of similar licensing agreements between MNOs, or between MNOs and multiple system operators (MSOs), shows that such agreements are laboriously negotiated, bilateral in nature, and inflexible in their offerings.

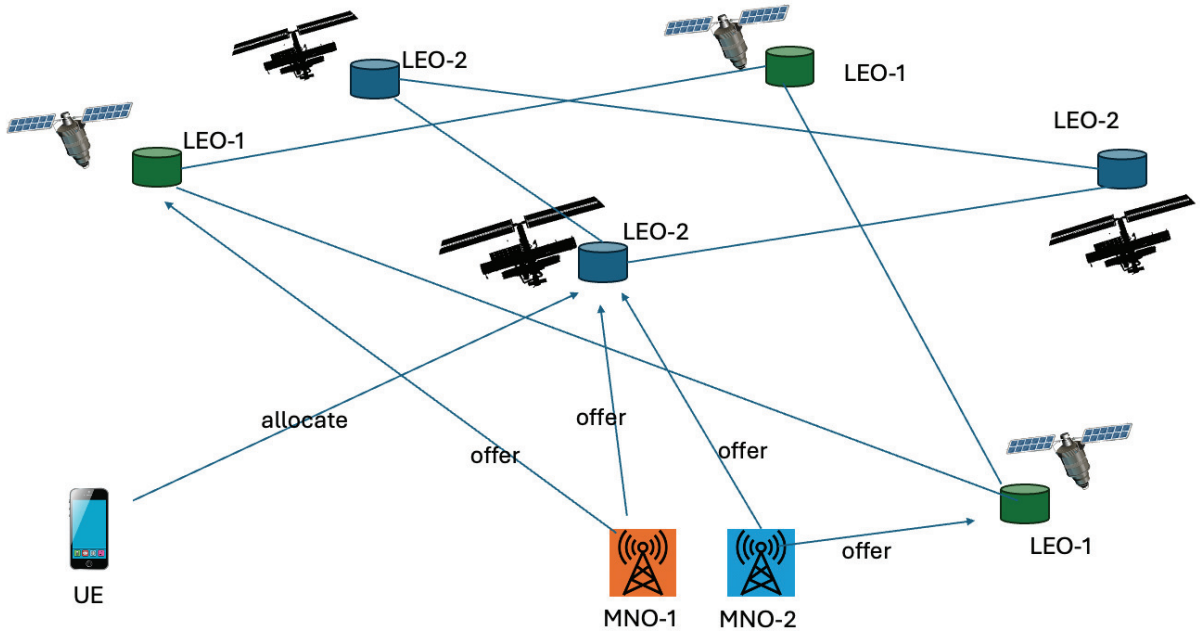

Fig 1: MNO / SNO Low Earth Orbit (LEO) Market

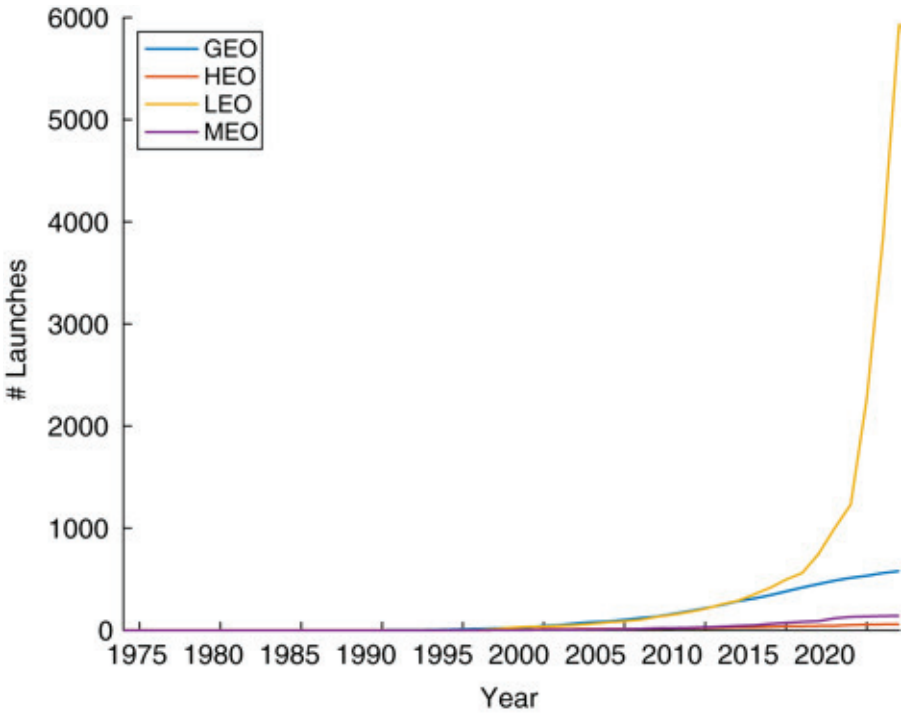

Fig 2: LEO Deployment trend from [5]

Known Prior Art

Blockchain and distributed ledger technology is mature and readily available as open source, e.g., Hyperledger Fabric (https://hyperledger-fabric.readthedocs.io/en/latest/). The state-of-the-art transaction processing algorithms used for contractual integrity (gossip, leader election, consensus) were designed for enterprise datacenters and are not appropriate for highly dynamic orbital deployments [8].

A similar SNO/MNO market was proposed in "Democratizing {Direct-to-Cell} Low Earth Orbit Satellite Networks" by Lixin Liu et al. [2], but it does not provide any smart contracts or any means to host core data services such as a ledger on-board (satellites), which is key to the present disclosure.

SUMMARY

The present disclosure describes a "Cloud-in-the-Sky," or geo-aware satellite on-board data infrastructure services. It is one aspect of the invention to provide a blockchain-based distributed ledger infrastructure to leverage the unique properties of Low-Earth Orbit (LEO) constellations and offer on-board (i.e., on the satellite) transaction processing for smart contracts.

Embodiments of the present disclosure can be used in mobile network operator ("MNO") LEO service provisioning.

One advantage of embodiments of the present disclosure is the ability to execute transactions on-board satellites to avoid the extra round-trip latency to earth-based services. This disclosure also takes advantage of the orbital movements to minimize communication and computational cost, as well as to make consensus and synchronization for transactional integrity efficient.

One particular use case described herein, although others are contemplated, is contracts between MNOs and satellite network operators (SNOs) to share the cost (and revenue) of offering high-speed direct-to-cell satellite connectivity. Note that although one focus is on a smart contract between a terrestrial MNO and the satellite network operator, the infrastructure described in the present disclosure is applicable to any smart contract that may need to be run, maintained, or executed on board the satellites operated by the SNO.

Given the rapid orbital movement of LEO satellites, a single connected user device is only served by the same satellite for about 5 minutes. This means that establishing and handing off a connection (to an overhead satellite so that the user device can maintain internet access through the MNO's network) needs to incur a low time overhead. Hence the proposal to run the smart contract service on board the satellites themselves.

While this proposal should reduce the time to establish a connection, it also comes with several infrastructure challenges, such as energy consumption, computation efficiency and transactional integrity. Existing underlying algorithms like leader election and consensus establishment were designed for data center deployment, not for a dynamic network such as an orbital constellation where the topology is in constant motion. At the same time there is an efficiency issue of orbital provisioning, where many satellites are under-utilized over low-population areas, such as oceans.

The basic idea, according to one embodiment of the present disclosure, is to designate some low-traffic geographic grid patches such as areas over oceans and low-population lands as service areas (SA). Satellites will only run the provided data infrastructure services as they pass over the SAs. This means that the cluster of nodes (e.g., satellites) running the service is in constant change and nodes entering the area will need to sync their data before they take part in the protocols.

A LEO deployment is organized by “satellites” moving in a circular orbit around the earth at a certain altitude and inclination angle (angle at which the satellite path crosses the equator). An “orbital plane” comprises a set of satellites on the same path. A “shell” is a set of orbital planes with the same altitude and inclination. Finally, a “constellation” is a set of shells. In this disclosure, these terms have their ordinary meaning in the art, as further defined and described herein.

Each satellite can have four (4) Inter Satellite Links (ISL), connected through optical links at free space speed of light to satellites within the same orbit (e.g., EAST, WEST) and neighboring orbits (e.g., NORTH, SOUTH). The network is furthermore bent around the edges in both the NORTH-SOUTH (columns) and EAST-WEST (rows) directions in what is referred to as a +GRID layout or 2D torus (doughnut shape).

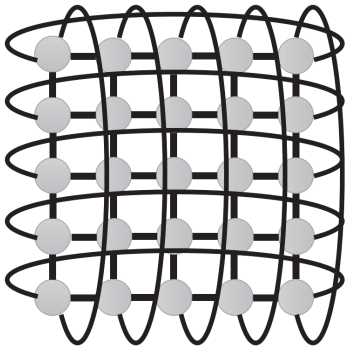

Fig 3: Satellite Shell 2D Torus

Each row and each column have the same number of satellites, while the number of rows and the number of columns may differ. Satellites in the same row (orbital plane) are spaced uniformly along the path (at equal distance from each other). The ISL communication within an orbital plane (EAST-WEST) is static in latency (and distance), but the latency varies between orbital planes (NORTH-WEST) over time, although predictably and periodically. And while there is no direct ISL communication available between shells in a constellation with this architecture. such routing may be possible through dedicated communication links or earth-based links.

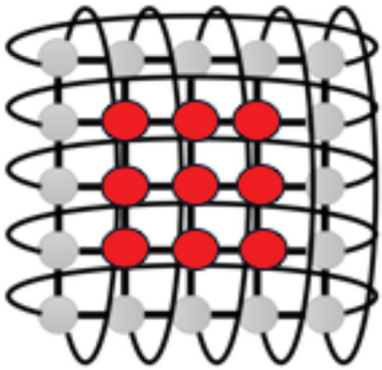

Fig 4: Service Area patch (red)

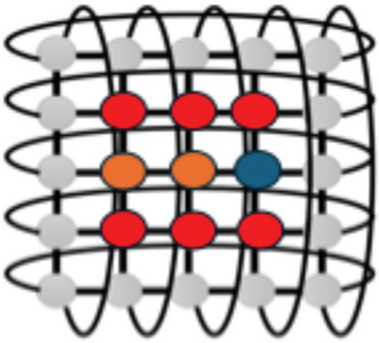

Fig 5: Leader row (orange) and leader (blue)

Within a service area (e.g., patch in shell) we define a row dedicated as the leader row, meaning a protocol leader is elected from this row.

Gossip Protocol and Routing

The basic broadcast protocol follows a gossip mechanism where each receiving node in the service area forwards messages not previously seen in the three directions the message did not come from. If a message is received that has previously been seen no further propagation is done by that node. Gossip messages may be started outside the service area but until it reaches the service area, only a single direction towards the shortest path to the service area will be used

to route the message. There could be several service areas in the same shell but there is no direct communication between the service areas and a message may only be directed towards a single service area at a time. Service areas may be used for sharding to split keys using a (consistent) hashing protocol towards a dedicated service area.

The gossip protocol does not rely on the torus wrap-around architecture for communication but will be more efficient with it. Similarly, the service area where communication will be spread does not have to follow a perfect square and non-contiguous regions may be more likely to form in case of failures when not. However, the leader row should be the orbital plane that covers the longest stretch over the area. (See next paragraph.) As long as a leader can be selected from the leader row and a majority of the service area nodes are operating, transactions should be possible to process.

Leader Election

To ensure transactional consistency a single node needs to be dedicated at any given time to serve as a leader within a service area hosting a ledger. Instead of fixing a single satellite directly to serve as leader, which is impossible since the nodes in the service area change over time, we will dedicate a row within the service area to serve as the leader row. Messages for the leader will be routed to the leader row and the leader row nodes will then be responsible for routing the message to the current leader. The leader should change as infrequently as possible and, hence, the leader election involves picking the node in the leader row that has the longest expected time left in the service area. Once that node leaves the service area it will re-assign leadership to the node that, at that point, has the longest expected time left in the service area (e.g., the node furthest east). Alternatively, the leader row may run a *ringleader-election* protocol with the priority being the expected time left in the service area.

Nodes who need to transmit messages to a leader do not need to know who the leader is. They only need to know the leader row (x-coordinate) and they can then send their messages south or north until the messages reaches the leader row. The node receiving the message in the leader row will know who the leader is and send it west or east if needed. Only the nodes active in a service area need to know who the leader is, so a node in the leader row but not in the service area can just forward the message to the service area, again, either west or east.

In order for a new leader in the leader row to accept the role of the leader instantly, they need top sync on the current last sequence number for transactions that the current leader set. When electing a new leader, the previous leader communicated this last sequence number to ensure the new leader picks up where the old leader left off. Any leader messages coming in to the old leader after the handoff will be routed to the new leader.

Node Migration

Nodes leaving the service area will stop keeping their ledger up-to-date, and nodes entering the service area will sync blocks with their neighbor immediately to the WEST before taking part in the transaction processing described below. A node entering the service area will request the most recent state of the neighbor to the west as well as all blocks on the blockchain that the west neighbor may have that the node does not have in its ledger. Since all blocks are indexed it is just a matter of requesting blocks beyond the local max index. If the sync fails because the neighbor to the west is not available the node may request an indirect sync with the north or south neighbor, assuming they are in the service area, to have their west neighbor provide the latest state and missing blocks since the last orbital cycle.

Service Area Example

Let's say users want to have a service area over the Atlantic. Given that the circumference of the Earth is 40,000 km, and the shortest distance over the Atlantic between Europe and the U.S. is 6,400 km, a typical time of LEO full-circle orbit is 90 min. Thus, the time a node spends in the service area would be roughly 6,400/40,000*90 = 14 minutes and 24 seconds. With about 40 satellites per orbit in a row and 17 planes in the service area it can host 680 ledger nodes, which is more than enough for a large-scale blockchain. This will ensure that migrations (in east-most column) or leader switches (in leader row) will not cause infeasible levels of overhead.

Transaction Processing

A transaction is a series of read and write operations on a ledger. It is performed in three major phases: (1) Transaction Execution, (2) Transaction Ordering, and (3) Transaction Verification and Commitment.

1) Transaction Execution

Any service area node with an up-to-date ledger may execute a transaction in this step. The transaction is executed against the local ledger as an initial verification of the validity of the transaction, but nothing is written to the ledger. All the read and write operations of the transactions are recorded with the keys read and written as well as their versions based on the local ledger.

2) Transaction Ordering

If the transaction is executed successfully locally the read and write lists are sent to the leader through the nearest leader row (e.g., straight NORTH or SOUTH). The leader will collect and serialize transactions and bundle them in blocks. The blocks of ordered transactions with unique increasing sequence numbers are then broadcast to all nodes in the service area using the gossip protocol described above.

3) Transaction Commitment and Verification

A node receiving a block will first inspect the sequence numbers to make sure all transactions are checked and verified in the order mandated by the leader, not the order of receiving the transaction. Once ordered by sequence numbers the service area node verifies the transactions are valid based on read and write traces and the current state for the given smart contract. The valid transactions will then be written to the local ledger and the state save to the local contract state. On successful commitment the node may notify the original transaction requester that a transaction completed successfully so that a consensus read/write can be performed, e.g., a write or read is only considered successful if at least a majority of all nodes in a service successfully wrote(read) a key. The block in the blockchain will also be verified to be consistent across writers for the transaction to be considered a success. Similarly, a failed transaction could also be reported to the original transaction requester. Note this mechanism is akin to the general *virtual synchrony* replication design. Transactions that fail to write to the ledger within a timeout may also be considered to have failed. All transactions are signed by the requester. Each block is hashed and the hash contains the hash of the previous block to ensure integrity of the ledger. This verification is done by each node in the service area before the block is written to the ledger.

Additionally periodic block syncing, e.g., triggered by the leader node, can be done in the service area where the latest block version numbers are gossiped to see if any node needs to pull new updates. This mechanism further limits the risk of failed transactions due to stale reads.

The latest blocks for each key will be kept in memory for faster reads and writes. Given that the Service Area nodes are all connected with speed-of light interconnects, a crashed node can resync from a neighbor as opposed to reading from permanent storage. Hence the service area can be seen as a distributed memory cluster.

The blocks contain the index, timestamp, transaction(s), and previous hash. The state contains the key value pairs the transactions operate on as well as the block index last written to the state. A transaction comprises a transaction id as well as a list of operations. An operation contains an operation type (read or write) as well as a version in case of read and a value in case of write.

An example state is shown below:


```
{
 "state": {
 "account_x": {
  "value": {
  "balance": 20
  },
  "version": 3
 },
 "account_y": {
  "value": {
  "balance": 10
  },
  "version": 3
 }
 },
 "block": 4
}
```


An example block in the chain for the state is shown below:


```
{
 "index": 4,
 "timestamp": "2024-09-18 15:05:52.435718",

 "content": {
 "ops": [
  {
```

```
  "key": "account_x",
  "op": "read",
  "version": 2
  },
  {
  "key": "account_x",
  "op": "write",
  "value": {
   "balance": 20
  }
  },
  {
  "key": "account_y",
  "op": "read",
  "version": 2
  },
  {
  "key": "account_y",
  "op": "write",
  "value": {
   "balance": 10
  }
  }
 ],
 "id": "4f0d0706-2390-4927-b91b-f3fc1fa10eda"
 },
 "previous_hash": "6070d7de1439abc124550deaca136fb71d1cc3f8bcc3ede0403cf2cfa4d2d9b2"
}
```


Smart Contract

A smart contract is set of methods that are all reading and writing state atomically. Each method is defined as a transaction comprising a series of read and write operations. The following is a smart contract implemented as a Python class.

```
class AccountContract(Contract):
 def __init__(self):
  pass

 def create(self,balance=0):
  self.transaction.write(str(uuid.uuid4()), {"balance": balance})

 def transfer(self,from_account=None, to_account=None, balance=0):
  value, _ = self.transaction.read(from_account)
```

```
    value["balance"] = value["balance"] - balance
    self.transaction.write(from_account, value)
    value, _ = self.transaction.read(to_account)
    value["balance"] = value["balance"] + balance
    self.transaction.write(to_account, value)
```

This contract is operating on local state and resulting transactions are sent to the leader node for ordering. The ordered transactions are then sent to all service area nodes for local verification and commitment to the local blockchain.

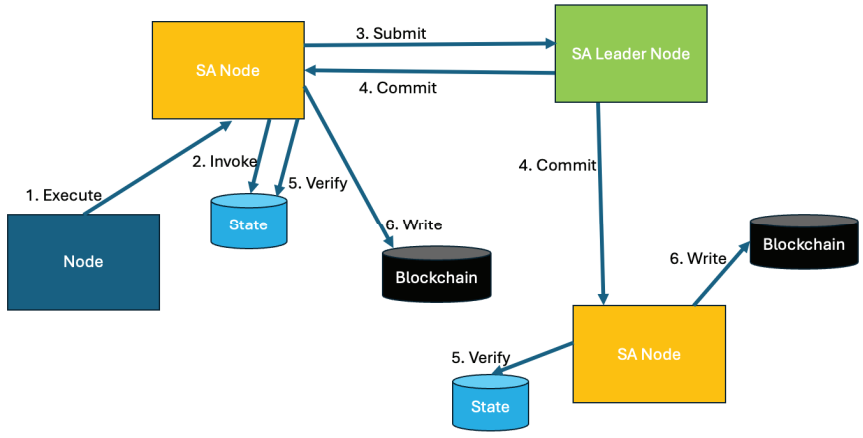

Fig 6: Contract Transaction Execution, Local Invocation, Ordering (Submit), and Local Verification and Writing to Blockchain

Distributed Databases

Any application running "in-the-sky" could use the proposed transaction processing mechanism herein to ensure eventually consistent and efficient processing of persistent state, with ultra-low-latency.

Related Work and Citations

[1] Pastukh, Alexander, Valery Tikhvinskiy, Svetlana Dymkova, and Oleg Varlamov. "Challenges of Using the L-Band and S-Band for Direct-to-Cellular Satellite 5G-6G NTN Systems." *Technologies* 11, no. 4 (2023): 110.

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

[3] Guidotti, Alessandro, Alessandro Vanelli-Coralli, Màrius Caus, Joan Bas, Giulio Colavolpe, Tommaso Foggi, Stefano Cioni, Andrea Modenini, and Daniele Tarchi. "Satellite-enabled LTE systems in LEO constellations." In *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 876-881. IEEE, 2017.

[4] Sandholm, Thomas, and Sayandev Mukherjee. "Smart Contracts for Mobile Network Operator Bandwidth Sharing." *Distributed Ledger Technologies: Research and Practice* 2, no. 4 (2023): 1-8.

[5] Vanelli-Coralli, et. al., “5G Non-Terrestrial Networks: Technology, Standards, and System Design,” IEEE Press 2024.

[6] https://www.fcc.gov/document/fcc-proposes-framework-facilitate-supplemental-coverage-space-0.

[7] Pfandzelter, Tobias, and David Bermbach. "Qos-aware resource placement for leo satellite edge computing." In *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*, pp. 66-72. IEEE, 2022.

[8] https://hyperledger-fabric.readthedocs.io/en/release-2.2/txflow.html.

APPENDIX

A Cloud in the Sky: Geo-Aware On-board Data Services for LEO Satellites

Thomas Sandholm, Sayandev Mukherjee, Bernardo A. Huberman

Next-Gen Systems, CableLabs, Santa Clara, CA

October 8, 2024

Abstract

We propose an architecture with accompanying protocol for on-board satellite data infrastructure designed for Low Earth Orbit (LEO) constellations offering communication services, such as direct-to-cell connectivity. Our design leverages the unused or under-used computing and communication resources of LEO satellites that are orbiting over uninhabited parts of the earth, like the oceans. We show how blockchain-backed distributed transactions can be run efficiently on this architecture to offer smart contract services.

A key aspect of the proposed architecture that sets it apart from other blockchain systems is that migration of the ledger is not done solely to recover from failures. Rather, migration is also performed periodically and continuously as the satellites circle around in their orbits and enter and leave the blockchain service area.

We show in simulations how message and blockchain processing overhead can be contained using different sizes of dynamic geo-aware service areas.

1 Introduction

With more connected devices and improved capabilities of connected devices, the need for high-bandwidth connectivity anywhere and anytime keeps growing at an exponential rate. High-population areas usually see investments in high-capacity communication infrastructure such as cell towers, small-cell antennas, radio heads and base stations. For an end-user device wanting access to a wireless mobile network, having low-latency access to this infrastructure not only allows the initial connection to be established faster, but also encounters fewer restrictions on the effective throughput, given timeouts of various acknowledgments in the protocols.

For remote or sparsely populated areas not served by the communication infrastructure described above, satellite communication is an attractive, albeit expensive, solution. Recent improvements in antenna technology as well as satellite launch economics have contributed to a 12-fold increase in the number of Low-Earth-Orbit (LEO) satellite launches in the last decade [27]. Because LEO satellites orbit the earth at a lower (and fixed) altitude they are ideal for providing communication services.

Current Non-Terrestrial-Networks (NTN) follow a *bent-pipe* architecture whereby the satellites serve as a simple relay between communicating parties on earth (the user device and a terrestrial base station, say). As a result, two-way communication between a target device and a base station providing network services result in messages traveling between the LEO satellite and earth four times, severely impacting both the latency and bandwidth the networks can provide. To overcome this overhead, recent efforts to move some core network services on-board the satellites have gained in popularity. The resulting architecture is often referred to as a non-transparent or regenerative, indicating that the satellite takes an active part in the communication beyond just relaying between parties on earth.

Apart from latency and bandwidth concerns, NTN networks also have to address the challenge of spectrum access. As LEO satellites circle the earth they need to make sure their transmissions do not interfere with (typically nationally licensed) terrestrial networks anywhere in their orbit, while at the same time being easily accessible to standard communication devices, such as cell phones. This has lead to a natural collaboration between Mobile Network Operators (MNOs) and Satellite Network Operators (SNOs). While a LEO satellite can circle the earth in 90 minutes, its orbital arc typically allows it to serve a given earth-located station for only about 5 minutes. Thus, any connectivity between a user device and a terrestrial base station through an LEO satellite needs to be established efficiently, depending on the current local constraints and spectrum availability.

We propose to host a bandwidth ledger that enables on-demand purchases of cellular bandwidth akin to the way compute resources are purchased in the Cloud, and that allows revenue sharing between the MNOs and SNOs. Given the latency concerns described above, we further propose to host this ledger entirely on-board the LEO satellites, using their Inter-Satellite-Link (ISL) free-space optical communication links.

In Section 2, we summarize the principal contributions of the present work. We then provide an overview in Section 3 of related work, discussing the differences between our approach and those of other researchers. In Sections 4 and 5 we provide a quick background overview of the two main technologies that are part of our proposed solution, namely Blockchains and LEO satellite constellations. The main section describing our proposed protocol is Section 6, followed by Section 7 describing a simulator and visualization tool for on-board blockchain processing in LEO constellations. We present the results of evaluation of the proposed protocol in Section 8 and our conclusions in Section 9.

2 Contributions of the present work

Note that communication between satellites is literally happening at the speed of light, but computational overhead can be an issue, in particularly with re-generative workloads or multi-tenant radio heads serving many MNO spectrum bands (e.g. on-board gNodeBs). Moreover, adding capacity to these networks is costly, as the circular motion and equal spacing means that there are the same number of satellites serving low-traffic areas such as big oceans and deserts, as the most populous areas such as metropolitan cities.

Our solution addresses all these challenges by running a distributed ledger to execute smart contracts using the unique structure of LEO ISL communication as well as the geographic properties of orbital cycles.

We propose two data infrastructure primitives: a *gossip protocol*, and a *distributed transaction*

processing protocol, that are customized to be efficient in LEO communication ISL networks and can be used to implement smart contracts as well as distributed databases for core communication services to avoid a round-trip to ground stations. We introduce the notion of a Service Area (SA) which denotes a geographical area where data services will be actively hosted. We further introduce the novel concepts of *leader row* and *neighbor migration* to support moving the cluster of active participating satellites in and out of the service area as they move around in their orbits.

3 Related Work

The many challenges of offering cellular connectivity from satellites, including doppler effects and latency issues, are described in [8]. These and other challenges in using the traditional satellite communication spectrum in the L-Band and S-Band for direct-to-cell connectivity [16] are the reason for SNOs and MNOs to coordinate their usage and allocation of spectrum resources. FCC has also recognized the value of re-using MNO spectrum for extended or supplemental coverage through satellites[1].

Previously, we have designed smart contracts for more efficient spectrum sharing among terrestrial MNOs [23, 21, 22, 20]. The present work extends this idea to LEO-based networks.

A similar approach of micro-contracts between MNOs and LEO SNOs is described in [12], where the authors propose an architecture for giving MNO customers easy access to SNO-provided multi-tenant direct-to-cell networks via standard SIM cards. Customers purchase tokens to use satellite services from their MNO, which will be written into trusted hardware on their SIM cards. When satellite access is needed, the SNO will be able to verify the validity of the token as well as mark it as used before providing service.

We have previously envisioned a similar architecture for ad-hoc multi-provider bandwidth purchases using eSIMs and an open distributed ledger market in [23] and [21] to overcome the inefficiencies of establishing new roaming contracts between providers. The advantage of our approach compared to that of [12] is that contracts do not need to be negotiated ahead of time and there is no need to keep tokens on SIM cards. However, our approach requires the presence of a distributed ledger accessible to both the end-user device and the mobile operator, and thus the performance of the system depends on the placement and migration of the distributed ledger.

QoS optimization is addressed via edge-service placement in LEO constellations in [17], where the authors select an optimal subset of satellites to host a specific service like a CDN or IoT data processor. In the scenario they address, the primary benefit is cost as only a subset of satellites deemed optimal to host services will be equipped with the necessary hardware. In contrast, in the present work we propose a software allocation solution that reduces the load on satellites to allow lower capacity hardware to be provisioned on all satellites. Moreover, our software placement is aware of the geographic position of the satellites in order to exploit idle resources on satellites not actively engaged in communication services. Our focus is less on reducing hops and instead on limiting broadcast overhead while still getting the benefit of replication. We note, however, that we chose to adopt their +GRID 2d torus model of ISL communication for our work as well.

Geo-aware LEO ISL routing schemes are investigated in [18]. The key to their approach is to embed geographical information in the MAC addresses of the communicating terminals to route packets more effectively without having to change the allocated IP addresses. With this approach

[1] https://www.fcc.gov/document/fcc-proposes-framework-facilitate-supplemental-coverage-space-0

they can better handle handovers and reduce delays in the dynamic and constantly moving network conditions of LEO satellites. Our focus is more on data infrastructure service provisioning and load balancing while taking geo-position into account, as opposed to routing.

ISL routing is also considered in [26], where the main goal is to effectively find alternative paths in case of failure while making sure that the link capacity in the network as a whole is optimized. They argue that path restoration instead of link restoration is a more efficient way of recovering from link failures. We only indirectly deal with failures in that we process transaction on a large set of satellites concurrently and in virtual synchrony, so that reaching any of these satellites would allow access to the up-to-date data with eventual consistency guarantees.

There have been many efforts to simulate satellite communication in software, e.g. [24] and [5]. We opted to build our own simulator to focus on the parts that mattered to our study, namely the LEO ISL links and the service migration and movement of satellites, as well as the ability to run custom software on each satellite node. The way we implement orbital planes as processes and satellite nodes as threads allows us to simulate large constellations effectively while easily facilitating and monitoring all communication between nodes. Furthermore, our simulator is written entirely in Python3 with only a couple of external dependencies for inter-process communication (HTTP REST). The visualization is written in standard JavaScript HTML5 running on a local Web server, which makes it easy and quick to set up and run locally on any laptop, as well as to demo remotely.

4 Blockchains and Distributed Ledgers

Blockchain technology was popularized with the virtual bitcoin currency [14] and provides a way to maintain a distributed ledger consistently and scalably across a large number of nodes. The 2-phase-commit transaction protocols used in traditional relational databases [2], for instance, scale very poorly as the transaction as a whole fails if only one party maintaining state fails. Blockchains originally provided a way to ensure consensus by what is referred to Proof of Work (PoW) where only parties that solved complex math problems were allowed to write into the ledger. The way a blockchain serializes blocks and maintains a hash of the previous block in the hash of the current block, it is easy to validate the internal consistency of the chain. PoW blockchains may contain competing forks of the blockchain but the longest chain wins if there is a conflict. These types of blockchains are appropriate in environments where anyone is allowed to write into the ledger in a large community of untrusted parties. The computation to solve math problems is however energy hungry and since LEO satellites have a limited lifetime depending on how long their battery lasts, this type of blockchain, referred to as permissionless, is not appropriate for our use case.

Instead we focus on permissioned blockchains that only allow authenticated and trusted participants to write into the ledger. Hence the blockchain can never fork into competing branches and no work is wasted. Some nodes may have fewer blocks but they will eventually catch up. Still the protocol needs to deal with failures and needs to ensure a consistent ordering of transactions. This is typically done by having a single node act as the leader to provide consistent ordering. Electing a leader is a critical part of the protocol as having no or multiple leaders will cause the system to fail as a whole. Leader election can be done by various consensus algorithms such as Paxos [11] or Raft [15]. As we shall see later we can provide a more suited and simpler leader election algorithm for the LEO case that is similar to the token ring leader election algorithm [25].

Permissioned blockchains like Hyperledger Fabric[2] go through the following high level steps to process a transaction: (1) execute the transaction against local state to ensure validity and record all versions read and what is written in each atomic unit (transaction), (2) send the read and write operations to the current leader to order all transactions globally, (3) once a sequence number is attached to each transaction or optionally they have been bundle in larger blocks broadcast them back (via a gossip protocol) to all nodes maintaining the blockchain state, (4) nodes receiving the transactions will validate and execute them in the order given to a local blockchain and state repository. Validation ensures that the version read has not been updated by some other transaction before writing to the same state.

As we shall see later we follow the same high-level steps which could be compared to the virtual synchrony [3] state replication approach, but with customizations at each step.

This process allows writes and reads to be done with eventual consistency [28], or optimistic locking as opposed to with strict atomicity, consistency, isolation and durability (ACID) guarantees, known from protocols such as two-phase commit [2]. As previously mentioned, the issue with the ACID protocols is that they scale poorly across large sets of replicas as failures become more likely.

A blockchain could in theory be implemented on a single node to ensure consistency but that would also reduce availability, so most deployments have the architecture of a distributed ledger with eventual consistency guarantees. This is also the model we follow.

Eventual consistency (guarantees) can be defined as follows. Assuming there are n nodes in the service area, if w nodes acknowledge they committed the transaction, and r nodes are used to read the data, then consistency can be guaranteed iff $w + r > n$ (see [28]). The tradoff follows from relaxing the C in the CAP theorem [4].

5 LEO Constellations

Low-Earth-Orbit (LEO) satellites are typically orbiting at a fixed altitude above the earth in the range of 311 to 621 miles [6]. In contrast to GEO satellites that follow the rotation of the earth to appear at a fixed point from a given position on earth, the LEO satellites rotate faster than the earth spins. The velocity and thus orbital period depends on the altitude of the satellite and mandates the number of satellites needed for full coverage. As an example, a LEO satellite at 391 miles altitude orbits the earth in about 97 minutes. A typical constellation at this altitude designed to provide full coverage with 34 satellites in each orbital path [17] results in a new satellite appearing over a fixed point on earth every 3 minutes.

Satellites on the same orbital path are referred to as an orbital plane. They are placed at equal distance from each other with a wraparound, so the first satellites is at the same distance from the second as from the last. These inter-satellite distances do not change over time. Orbital planes are further organized into shells or constellations[3] where each plane has the same altitude over earth and inclination angle[4]. Communication across orbital planes are possible but the distance to the

[2] `https://www.lfdecentralizedtrust.org/projects/fabric`

[3] Sometimes constellations refer to a set of shells but we only assume intra-shell communication is available here so use constellation and shell interchangeably.

[4] angle at which its path crosses the equator

nearest neighbor in a different plane may differ over time, although it is predictable and cyclical. Hence, within plane communication is more reliable.

Each satellite would typically have 4 inter-satellite links to neighboring satellites, e.g. west, east, north and south, using free-space optical connections to transmit at the speed of light. The network in a shell can thus be seen as a connected mesh forming a 2d-torus where there is a wrap-around both in rows and columns (see Figure 1). This architecture is referred to as a +GRID inter-satellite link (ISL) network [17], and it is the one we assume for our work.

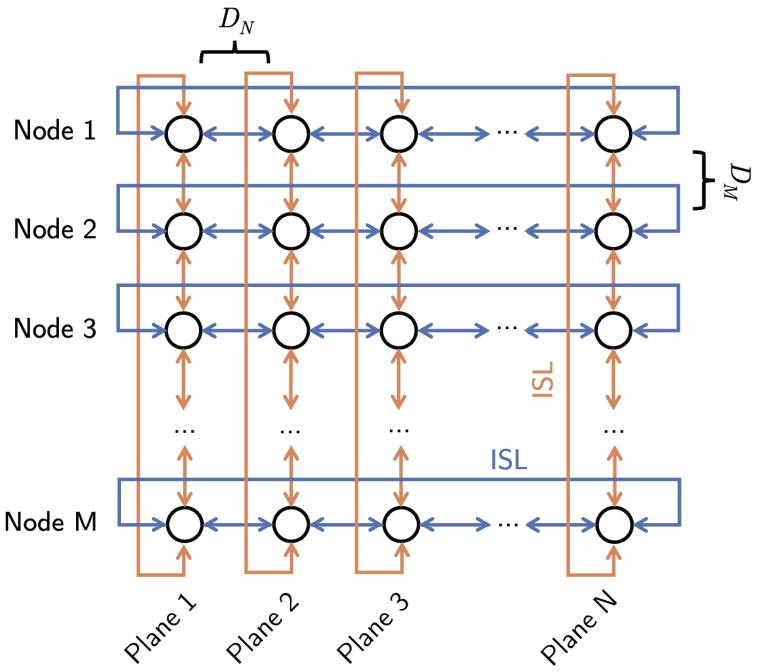

Figure 1: +GRID 2d torus ISL. Source: Pfandzelter and Bermbach [17].

As described in Section 2, a service area (SA) is defined as an area where satellites are typically less loaded with communication workloads, such as over the oceans. Any satellite passing over this area will take an active part in hosting the data services.

For example, the Pacific Ocean stretches about $15,500$km from the Arctic to the Southern Ocean and $19,800$km from Indonesia to the coast of Colombia, which is close to half the circumference of the earth. So up to a third of the planes and half of all the satellites in a plane could hover over the Pacific Ocean at any given time. A typical LEO constellation would make up about 22-28 orbital planes with 5-72 satellites per plane or 375-1584 satellites in total[5]. So about 8 planes and 39 satellites per plane or about 321 satellites could be wasting their capacity while idling over the Pacific. Having 300+ nodes in a distributed ledger with virtually unlimited speeds to interconnect them is clearly a resource worth exploiting.

One major issue with these networks is that adding more capacity by sending up more satellites to increase coverage also means that there is more waste in terms of periods where the satellites

[5]These numbers are based on the data in Table I in [17]

are idle. Furthermore, these periods are predictable and cyclical so easy to identify and exploit without any central coordination[6], which is the key idea behind our approach, described next.

6 Protocol

Next, we discuss the different parts of our protocol: gossip, transaction processing, migration and smart contracts.

All our protocols rely on the 2d-torus architecture of LEO ISL. Furthermore we define a blockchain Service Area (SA) as a block of contiguous nodes (each node being hosted on a satellite) over some geographic area with low terrestrial traffic load, such as the Pacific Ocean. All satellites in the constellation may communicate with each other through the one-hop architecture of the 4 ISL links (see Figure 1), but broadcasting is only done in the service area, and only nodes in the service area take active part in processing transactions and maintaining up-to-date blockchain state. Owing to the orbital movement, the set of nodes that are in the service area keeps changing, so we need to do continuous migrations of the blockchain ledger to nodes that move into the service area. Note that the nodes currently in the service area do not need to be informed about the new nodes moving into the service area, because the movement of every satellite is completely predictable and so each node in the service area can independently determine when it will move out of the service area and which node will enter to take its place.

6.1 Gossip and Routing

Updates on one node need to be broadcast efficiently to other nodes in the blockchain cluster to ensure the window of inconsistency is kept small. At the same time a single path to distribute updates can be a single point of failure, e.g. if all updates go through a central node. So, for reliability most blockchain systems implement some form of gossip protocol [7]. These protocols are modeled after epidemic virus infection spread in a human population: if infected, you then infect your nearest (uninfected) neighbors who go on to infect their (uninfected) neighbors, while an encounter between two infected people does not propagate the virus further. Typically a handful of neighbor nodes are configured to propagate gossip messages to. Care is taken not to propagate to a neighbor that sent you the broadcast message. A message that has already been seen, e.g. previously received by a different neighbor, is not propagated again.

The gossip protocol is also efficient, in the sense that N nodes can be synchronized in $O(\log N)$ iterations despite node failures and data losses in transmission [7]. The gossip protocol on a torus (which is the scenario that matches LEO satellite constellations) was analyzed in [13]. Improved gossip protocols specifically for use in blockchains were proposed and evaluated in [1], [10], and [19].

In a LEO constellation architecture following the +GRID ISL structure the neighbors are simply the north, south, east, and west connections. If a message comes from one direction it is propagated into the other three, and previously seen messages are dropped. Furthermore we restrict propagation to within the service area we defined, typically a grid but it could have any shape as long as all the nodes in the service area are contiguous. All messages traversed also receive

[6] each satellite node can determine independently when they are above a certain geographic area without a need for external communication

the gossip due to the one-hop communication structure, ensuring the entire service area receives a broadcast efficiently.

Communication within an orbital plane tends to be more reliable and thus more nodes are reached within a time unit if west/east spreading is prioritized but propagation can be done concurrently in all directions.

In some cases a node outside the service area may want to trigger a broadcast within the service area or simply execute an operation on any node that is closest within the one-hop structure and at the same time is currently in the active service area. Depending on where the broadcasting node is located and how the service area is configured the shortest path to a target could be in the opposite direction using the wrap-around nodes of the 2d-torus.

For instance a transaction may be created from any node but needs to be executed in a node in the current service area and the leader node that orders transactions then broadcasts the ordered transactions within the service area only.

Changing the leader node could lead to disruption in transaction processing and hence we want to do that as infrequently as possible. Thus a leader is not dropped until it leaves the service area. The leader's identity is also broadcast through the gossip protocol within the service area. To optimize communication we limit the nodes that can be leaders to a single row, the *leader row*, i.e., the nodes in a selected orbital plane that are within the service area. In theory, it is sufficient for only the leader row nodes to know the identity of the leader, as all other nodes could send messages to the leader via the nodes on the leader row. The fewer the nodes that need to know who the present leader is, the smaller the window of downtime when a new leader is elected (details of the method of electing the new leader node are given in Section 6.3). After the new leader node has been elected but before the current leader node has exited the service area, if the current leader node receives any messages intended for the leader, it can simply forward them to the new leader node.

6.2 Transaction Processing

Transaction processing is at the core of our protocol as it is what ensures eventually consistent distributed states and availability even in the case of partial failures. A survey of blockchain consensus algorithms may be found in [9]. Distributing state across a large area becomes even more important in an ISL network where only single node hops are possible. As we have previously mentioned it is enough to contact any node (satellite) currently in the service area to read the current state.

State here is simply a key-value database where the key is a string and the value is an arbitrary object defined by the application, e.g., a JSON dictionary. The state is updated with transactions, a transaction being defined as an ordered list of atomic read and write operations on the state. Each such operation either succeeds or fails. Transactions can be submitted by any node in a constellation and executed by any node in the service area. However, the order in which transactions are executed is kept consistent across the constellation.

The ground truth of the order of all transactions is logged in a blockchain ledger, replicas of which are maintained by all the nodes in the service area. The content of a block in the blockchain is the read and write operations for the transaction as well as the hash of the previous block in the blockchain. Hence the validity of transactions can be independently verified by any node in the service area. The state is maintained separately but logs the block id of the last transaction that made an update to the state. Whenever a key in the state is written to, the version is bumped up

and all read operations in transactions note which version they read. Note that a transaction can span many keys in the state, e.g., read from one state key and write into another.

The processing of a transaction is described in full below.

1. A transaction is submitted by any node in the constellation. It is then routed via ISL to the nearest service area prioritizing within-plane hops. Note that if a service area is defined to span the full set of orbital planes this routing step only requires within-plane routing.

2. The node within the service area that receives the transaction executes it locally without writing to the ledger (blockchain) or state. This is done by reading and writing to a separate in-memory copy of the state to ensure the transaction is valid assuming: (a) the node's state is up to date, and (b) the transaction does not rely on state that has since been updated, i.e., there is no version mismatch between the read key and the key in the state of the node executing the transaction.

3. If local execution succeeded, the node forwards the transaction to the leader node for ordering.

4. The leader node attaches a sequence number (which is global to the constellation) and then broadcasts the transaction, or a set of ordered transactions, to all the nodes in the service area using the gossip protocol[7].

5. A node receiving the transaction broadcast will validate the transactions in sequence and write the transactions that succeeded to the blockchain as well as update the state and key version numbers accordingly. The same verification as in the local execution of the transaction is performed with the difference that the validation is done in an order mandated by the leader node as opposed to the time of arrival.

6. When submitting a transaction, a transaction ID is generated and the submitter may query any node in the service area to check whether a transaction completed successfully. A successfully completed transaction including all its read and write operations are written into the blockchain and the state is updated accordingly.

The messages passed during processing of a transaction can be seen in Figure 2.

6.3 Service Area Migration

They key aspect of our protocol that sets it apart from other blockchain systems is that migration is not only done to recover from failures but also done periodically and continuously as the satellites circle around in their orbits and enter and leave the service area. Therefore it needs to be very efficient and cause minimal disruption to transaction processing.

The migrations are complicated by the fact that not only do we need to make sure the state and the blockchain migrates properly but also that the leader role migrates as well, which includes migrating in-memory state of the old leader to the new leader, and the election of a leader using a consensus protocol.

[7]This is similar to the Ordering service process described at `https://hyperledger-fabric.readthedocs.io/en/release-2.2/txflow.html`

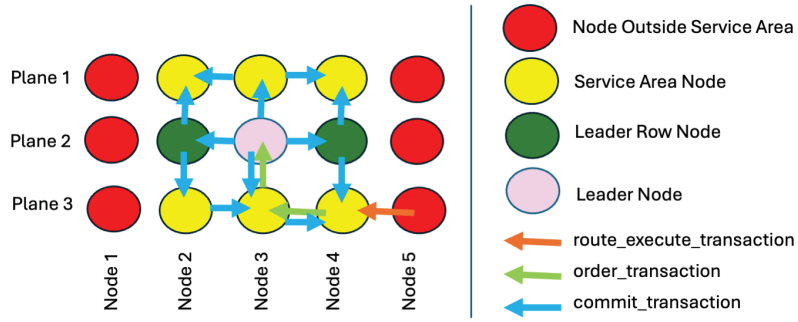

Figure 2: Transaction route execute, order, and commit (gossip) message passing.

As a set of nodes are about the leave a service area and another set is about to enter the service area the following steps, which we refer to as *neighbor migration*, are performed:

1. If the current leader is among the nodes that are leaving the service area, a new leader is elected. The old leader simply sends a message to the new leader that it should take over the role as leader. The leader election can only happen within a preconfigured orbital plane and thus the same row in the ISL torus. The new leader is the node furthest to the east in the leader row inside the service area. If that node does not respond the one west of it is elected and so on. A more formal ring leader election algorithm may also be executed. Once a new leader has been elected, the newly-elected leader broadcasts that it has taken up the role of leader within the service area using the gossip protocol.

2. Each node about to enter the service area synchronizes and updates their state and blockchain with their neighbor directly to the west already inside the service area. Only the blocks in the blockchain that were appended after the last rotation (of these nodes) in the service area need to be retrieved, and this communication is efficient as it is a single hop. Furthermore, all the nodes across all orbital planes that are about to enter the service area can do this migration concurrently and using different target nodes to synchronize from. This again makes the migration operation scalable and efficient.

3. Finally when the migration is complete the west-most nodes in the service area orbits can drop out, and the borders of the service area can be moved to include the nodes that just migrated into it. Note that the timing of these steps can be predicted within each node, based on the location of the nodes, and thus there is no need to send messages to trigger these steps.

In order for the new nodes (those just entering the service area) to execute transactions locally they need an up-to-date version of the state as well as the blockchain. For them to write into the ledger they also need to know the last known transaction sequence number so they can buffer incoming ordered transactions if they arrive out of order. Finally, if a new node is the new leader, it also needs to have an up-to-date version of the global sequence counter assigned to transactions that are to be ordered.

6.4 Smart Contracts

To be able to test our transaction processing infrastructure better, we also provide a smart contract programming construct that allows us to define smart contracts. A smart contract is simply an interface with methods that read and write from the state maintained in the blockchain, as well as define what the structure of the state is. Each method when executed generates a transaction comprising an ordered list of read and write operations that can be submitted into the constellation for processing and that will be written to the ledger with the eventually consistent guarantees.

Below is an example of a smart contract defining a bank account contract with the ability to transfer money between accounts.

```
class Contract:
    def call(self, contract, op, args):
      self.transaction = Transaction(contract=contract)
      getattr(self, op)(**args)
      return self.transaction

class AccountContract(Contract):
    def create(self,balance=0):
      self.transaction.write(str(uuid.uuid4()), {"balance": balance})
    def transfer(self,from_account=None, to_account=None, balance=0):
      value, _ = self.transaction.read(from_account)
      value["balance"] = value["balance"] - balance
      self.transaction.write(from_account, value)
      value, _ = self.transaction.read(to_account)
      value["balance"] = value["balance"] + balance
      self.transaction.write(to_account, value)

def register(name, clazz):
  contracts[name] = clazz

register("AccountContract",AccountContract)

## Example Usage:
## transaction = invoke_contract("AccountContract",
##                               "transfer",
##                               {'from_account': account1,
##                                'to_account': account2,
##                                'balance':2})
def invoke_contract(contract, op, args):
    return contracts[contract]().call(contract,op,args)
```

7 Simulation and Visualization

To be able to test transaction processing and smart contract execution in a constellation in motion we developed a simulation using Python and a Web visualization.

In the simulator we define ISL communication paths such that each node can only communicate with its immediate neighbors to the north, south, west and east. Each node has a (x, y) grid coordinate representing satellite x in an orbital plane y. A service area defines a range of x values and a range of y values that may be updated at any point in the simulation to account for orbital movements.

Since a constellation may have thousands of nodes that can all communicate with each other concurrently and independently, we designed the simulator to use as few resources as possible while still being able to run realistic transaction processing scenarios and scale up to large constel-

lations. Each orbital plane is implemented as a separate Python process that exposes a REST API to communicate with other orbital planes (in north-south links). Inside an orbital plane (west-east links) all communications are done with a thread pool. In general a good size of the thread pool is the number of nodes in the orbital plane, but it can also be set based on load.

Each node writes to its own copies of the state and blockchain, both of which are represented by JSON files in the simulator. When a node wants to send a message to another node it simply specifies the (x, y) coordinates of the target and the simulator will use a combination of one-hop messages within the thread pool and REST APIs to reach the target using the routing and gossip protocol defined previously.

At any time, a node can be asked to synchronize its state with its neighbor to the west. Similarly any node can be asked to assume the role of a leader. Nodes within the current service area can execute and validate transactions locally and submit validated transactions to the leader node for ordering. The leader node can sequence transactions and broadcast the sequenced transactions to all service area nodes using the gossip protocol described previously.

For evaluation purposes we define a smart contract that can create accounts and transfer money between accounts, as well as monitor the balance of each account as the satellites move around their orbits.

A web simulator shows a grid of the 2d-torus with the current state in each node for a given account. It also shows the current nodes in the service area (yellow), the nodes outside the service area (red), the nodes in the leader row (green) and the current leader (pink).

The web simulation moves all satellites one step west every time interval (configured to 10s). At any time, transactions may be executed, such as creating new accounts and transferring money between accounts. The service area follows the rotation of the earth to always reside above the Pacific Ocean.

We use a demo constellation of 4 orbital planes and 28 satellites per plane, where 6 satellites across all 4 planes cover the service area at any given time.

A screenshot of the visualization can be seen in Figure 3.

8 Communication Evaluation

We now take a closer look at the communication overhead of our solution. In particular we are interested in the total number of messages generated per transaction for different sizes of service areas. We use a 28×4 grid of total nodes, and always use all orbital planes but vary the range of satellites in a given plane (x-range) that are in a service area. Varying the x-range from 5 nodes to 10 nodes results in service areas having 20 to 40 nodes. We use 70 threads in each simulation process that represent an orbital plane.

The evaluation involves executing 3 transactions, 2 account creations and then a transfer between the newly created accounts for each period, or rotational position of the earth. We define 28 rotational positions, meaning that after 28 periods the satellites will be back in their original positions.

We keep the service area over the same geographic patch (Pacific Ocean) so that new nodes enter and leave the area with every rotation. For each configuration we do 10 full cycles of rotations around the earth and then measure the number of messages sent in the system. A message that is routed between nodes is counted as a new message for each hop, as all communication is single hop

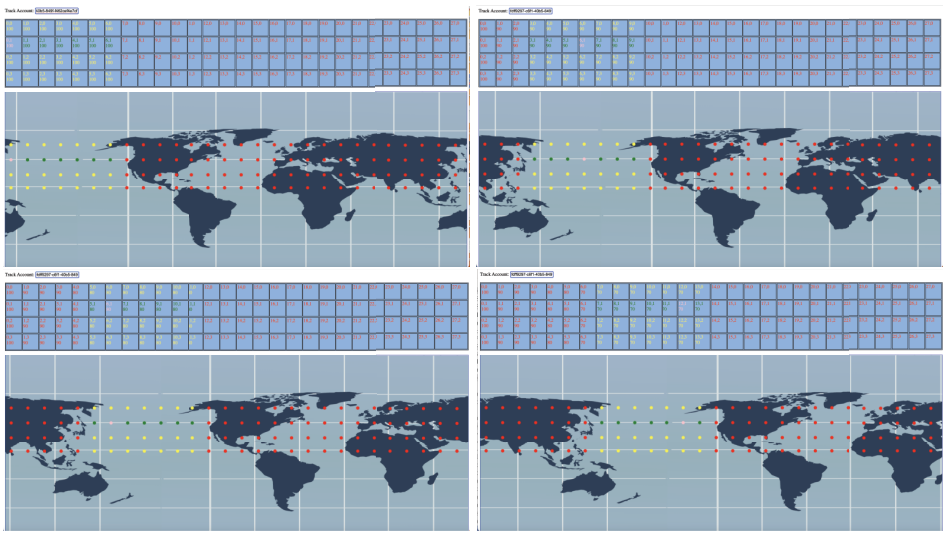

Figure 3: Simulation Web visualization screenshots of periods 1, 4, 6, and 8. Top: Torus cells, Bottom: Earth overlay. Red node: outside of service area. Yellow node: inside service area. Green node: leader row node. Pink node: leader.

according to the LEO ISL setup previously described. The evaluation configuration is summarized in Table 1.

Table 1: Evaluation Configuration.

Transactions	841
Migrations	1120
Earth Cycles	10
Cycle Periods	28
Service Area Nodes	$\{20, 24, 28, 32, 36, 40\}$
Transaction Commits	$\{16820, 20184, 23548, 26912, 30276, 33640\}$

In Figure 4 we observe that the number of messages grows linearly with the size of the service area and that the gossip messages dominate the communication overhead for transaction processing. We note that the transaction commits also grow linearly with service area nodes as per the virtual synchrony design. Hence, adjusting the size of the service area is an effective way to limit both communication and I/O processing overhead.

Table 2 shows the proportion of different message types, again highlighting the dominance of gossip messages. Only the gossip messages change based on service area size. The *route_execute* message is sent if the sending node cannot execute the request, e.g. it is not in a service area and is asked to execute a transaction. The *sync_blocks* message is a variant of this where a node is asked by another node to sync state with its neighbor because it is about to enter the service area. In a live deployment the node would know internally when it needs to synchronize state as it is aware of its orbital path. The *sync_state* message actually synchronizes the blockchain and its state as well as in-memory state with its neighbor to the west to be able to participate in transaction execution

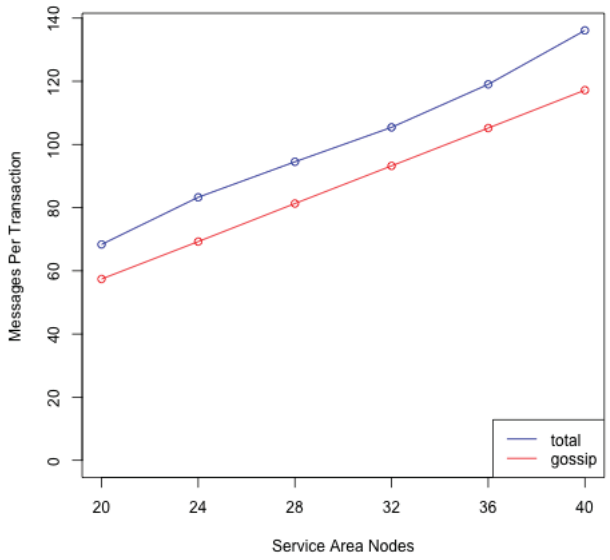

Figure 4: Total messages and gossip messages (for transaction commits).

within the service area.

Table 2: Message proportions in 7×4 grid service area.

Message	**Percent of total number of messages**
gossip	86%
route_execute	9%
execute_transactions	1%
order_transaction	1%
sync_blocks	1%
sync_state	1%

Figure 5 shows the gossip message distribution for satellites across orbital index and plane. The dip in plane 1 is because the leader originating gossip is in this plane (the leader row).

9 Conclusions

We have demonstrated how a blockchain can be hosted efficiently on-board LEO satellites to offer eventually consistent guarantees for distributed transactions and smart contracts. We believe that the importance and utility of such data infrastructure will increase in the future when satellite nodes are upgraded in compute and storage capacity to avoid expensive round-trip costs and to meet the stringent latency guarantees of 3GPP NTN non-transparent direct-to-cell communication.

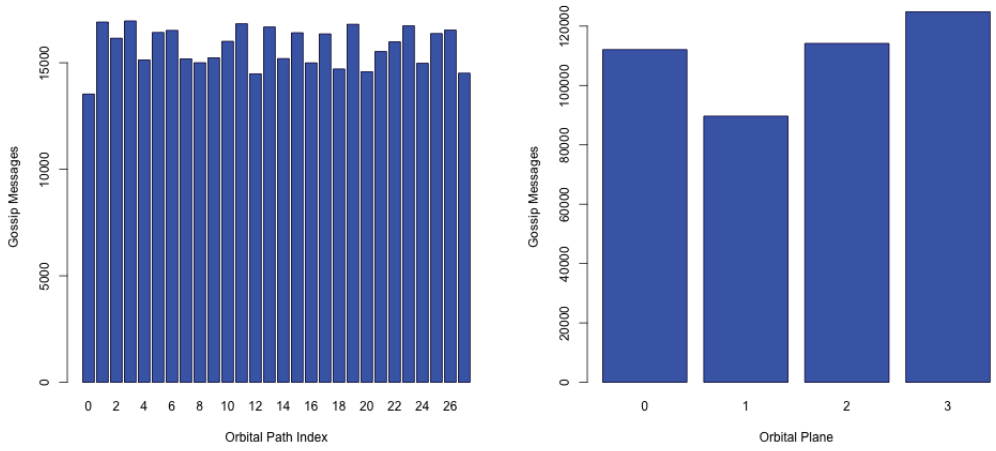

Figure 5: Gossip message distribution across orbital indices and orbital planes across all evaluation runs.

There are many use cases for the data infrastructure proposed and evaluated in this work. An example of an innovative application that uses the capability of such infrastructure is fast, on-demand authentication to a new MNO through an SNO via a smart bandwidth contract that automates roaming.

References

[1] N. Berendea, H. Mercier, E. Onica, and E. Riviere. Fair and Efficient Gossip in Hyperledger Fabric. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 190–200, Los Alamitos, CA, USA, dec 2020. IEEE Computer Society.

[2] Philip A Bernstein and Eric Newcomer. *Principles of transaction processing*. Morgan Kaufmann, 2009.

[3] Ken Birman and Thomas Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the eleventh ACM Symposium on Operating systems principles*, pages 123–138, 1987.

[4] Eric A Brewer. Towards robust distributed systems. In *PODC*, volume 7, pages 343–477. Portland, OR, 2000.

[5] Lin Cheng and Bernardo A Huberman. Auction-Based Efficient Communications in LEO Satellite Systems. *Available at SSRN 4286498*, 2022.

[6] Hervé Cottin, Julia Michelle Kotler, Daniela Billi, Charles Cockell, René Demets, Pascale Ehrenfreund, Andreas Elsaesser, Louis d'Hendecourt, Jack JWA van Loon, Zita Martins, et al. Space as a tool for astrobiology: review and recommendations for experimentations in Earth orbit and beyond. *Space Science Reviews*, 209:83–181, 2017.

[7] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, 1987.

[8] Alessandro Guidotti, Alessandro Vanelli-Coralli, Màrius Caus, Joan Bas, Giulio Colavolpe, Tommaso Foggi, Stefano Cioni, Andrea Modenini, and Daniele Tarchi. Satellite-enabled LTE systems in LEO constellations. In *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 876–881. IEEE, 2017.

[9] Ziad Hussein, May A. Salama, and Sahar A. El-Rahman. Evolution of blockchain consensus algorithms: a review on the latest milestones of blockchain consensus algorithms. *Cybersecurity*, 6(1):30, 2023.

[10] Kadir Korkmaz, Joachim Bruneau-Queyreix, Stéphane Delbruel, Sonia Ben Mokhtar, and Laurent Réveillère. In-depth analysis of the IDA-Gossip protocol. In *2022 IEEE 21st International Symposium on Network Computing and Applications (NCA)*, volume 21, pages 139–147, 2022.

[11] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, 2001.

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

[13] Ulrich Meyer and Jop F. Sibeyn. Oblivious Gossiping on Tori. *Journal of Algorithms*, 42(1):1–19, 2002.

[14] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *https://bitcoin.org/bitcoin.pdf*, 2008.

[15] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX annual technical conference (USENIX ATC 14)*, pages 305–319, 2014.

[16] Alexander Pastukh, Valery Tikhvinskiy, Svetlana Dymkova, and Oleg Varlamov. Challenges of Using the L-Band and S-Band for Direct-to-Cellular Satellite 5G-6G NTN Systems. *Technologies*, 11(4):110, 2023.

[17] Tobias Pfandzelter and David Bermbach. QoS-aware resource placement for LEO satellite edge computing. In *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*, pages 66–72. IEEE, 2022.

[18] Manuel Roth, Hartmut Brandt, and Hermann Bischl. Implementation of a geographical routing scheme for low earth orbiting satellite constellations using intersatellite links. *International Journal of Satellite Communications and Networking*, 39(1):92–107, 2021.

[19] Gokay Saldamli, Charit Upadhyay, Devika Jadhav, Rohit Shrishrimal, Bapugouda Patil, and Lo'ai Tawalbeh. Improved gossip protocol for blockchain applications. *Cluster Computing*, 25(3):1915–1926, 2022.

[20] Thomas Sandholm and Sayandev Mukherjee. A multi-armed bandit-based approach to mobile network provider selection. *arXiv preprint arXiv:2012.04755*, 2020.

[21] Thomas Sandholm and Sayandev Mukherjee. Bandcoin: Using smart contracts to automate mobile network bandwidth roaming agreements. *arXiv preprint arXiv:2104.02780*, 2021.

[22] Thomas Sandholm and Sayandev Mukherjee. SCNO: Smart city network operator. In *Proceedings of the 1st Workshop on Artificial Intelligence and Blockchain Technologies for Smart Cities with 6G*, pages 1–6, 2021.

[23] Thomas Sandholm and Sayandev Mukherjee. Smart Contracts for Mobile Network Operator Bandwidth Sharing. *Distributed Ledger Technologies: Research and Practice*, 2(4):1–8, 2023.

[24] Tim Schubert, Lars Wolf, and Ulf Kulau. ns-3-leo: Evaluation tool for satellite swarm communication protocols. *IEEE access*, 10:11527–11537, 2022.

[25] Mina Shirali, Abolfazl Haghighat Toroghi, and Mehdi Vojdani. Leader election algorithms: History and novel schemes. In *2008 Third International Conference on Convergence and Hybrid Information Technology*, volume 1, pages 1001–1006. IEEE, 2008.

[26] Jun Sun and Eytan Modiano. Capacity provisioning and failure recovery in mesh-torus networks with application to satellite constellations. In *Proceedings ISCC 2002 Seventh International Symposium on Computers and Communications*, pages 77–84. IEEE, 2002.

[27] Alessandro Vanelli-Coralli, Nicolas Chuberre, Mohamed El Jaafari, Alessandro Guidotti, and Gino Masini. 5G Non-Terrestrial Networks: Technologies, Standards, and System Design. 2024.

[28] Werner Vogels. Eventually consistent. *Communications of the ACM*, 52(1):40–44, 2009.