Leveraging Decentralized Networks to Locate and Identify LEO Debris

Jacob Kidd

Some other stuff goes here

Abstract. There are over 30,000 large (size greater than 10cm) and more than 1 million smaller micro-meteoroids and orbital debris (MMOD) in low earth orbit (LEO), or the region of space below 2,000 km [1]. According to NASA, orbital debris is the "predominate threat" in LEO [3]. These objects cultivate hazardous conditions for human activities in orbit, with the potential to impair or destruct satellites or human-occupied spacecraft, creating both the possibility of fiscal, infrastructural, and human loss. They often produce the need to execute expensive and time-sensitive maneuvers. Thus, the monitoring of debris objects is a vital activity both necessary to reduce risk, especially as enterprise and operations in orbit increase, and a cost-efficient endeavor, as it can reduce exertions or over-corrections by satellites in anticipation of a collision. The responsibility to identify and broadcast classification and orbital data is often shared between institutions, whether private, academic, or administrative in nature, worldwide, in many cases crossing geopolitical borders as a mutual duty and liability. However, to date, there exists no global registry or medium to circulate this information, and incomplete datasets continue to pose a threat. We propose an inclusive methodology and lightweight protocol to incentivize the production of corroborated, accurate datasets that model the positions and orbits of debris objects in LEO over a decentralized medium of global, agnostic data exchange.

1 Introduction

In the realm of space exploration, the proliferation of orbital debris presents a formidable challenge to satellite operations and future space endeavors. With over 23,000 pieces of debris larger than 10 cm currently tracked in orbit and millions of smaller fragments, accurate and timely monitoring of this debris is compulsory for ensuring the safety and sustainability of space activities (NASA Orbital Debris Program Office). However, conventional centralized approaches to aggregating observational data encounter significant hurdles, including scalability issues, data silos, and difficulties in incentivizing participation from diverse "stakeholders" with orbital interests. In response to these challenges, this paper proposes a pioneering solution: the establishment of a decentralized network for managing the aggregation of orbital debris observational data from global, distributed sources. By leveraging blockchain-based incentive mechanisms, a decentralized autonomous organization (DAO) model for protocol moderation, machine learning methods in the aggregation process, and a contemporary network architecture with greater feasibility for data management, this approach seeks to foster collaboration, transparency, accessibility, and data integrity within the space community. Through the collective efforts of distributed observers, we aim to create a reliable and comprehensive global registry of orbital debris object identification images that reflects continual consensus among contributing constituents, in the interest of facilitating informed decision-making and risk mitigation strategies in space exploration. This innovative approach aligns with the concept of the wisdom of crowds, wherein the collective input of diverse individuals leads to more accurate outcomes than that of any single expert (Surowiecki, 2004), and not only addresses the urgent need for improved debris

monitoring but also embodies the principles of international collaboration and technological innovation essential for advancing space sustainability in the 21st century.

2 Methodology

In this section, we will be examining the various methods, including instruments, used today to identify orbital debris in Section 2.1. In Section 2.2, we will investigate a method that could be used to represent as well as predict debris object locations . In Section 2.3, we put forth the data

2.1 Apparatus

Observation of orbital debris, comprising of defunct satellites, spent rocket stages, fragments from previous space missions, and micrometeoroids, is most commonly conducted by ground-based radars, laser brooms, optical telescopes, space-based telescopes, and examination of spacecraft that have experienced collisions and returned from space [4]. These instruments are primarily operated by institutional sources, such as NASA or the European Space Agency (ESA). Detection, for example, as carried out by the U.S. Space Surveillance Network, yields the identification of approximately 20,000 of these foreign objects [2], a factor of two thirds. Classification of debris orbit and specs is also important. According to NASA, objects have varying density and material (e.g. steel, aluminum, and plastic) [3], factors which figure into calculating auxiliary characterization, such as ballistic coefficients. For space operations, satellite collision avoidance, and maintaining the integrity of space infrastructure, this process is vital.

While feasible to occur in some measure, satellites themselves are frequently unable to identify the location of threatening debris, as objects may be outside the periphery of on-board visual monitoring instruments, or there exist no on-board instruments equipped with the task of identification. Some remotely-controlled satellites, however, actually specialize in capturing debris, for the purpose of recycling, using, for instance, nets and harpoons [13].

In the pursuit of decentralization, meaning promoting accessibility and reducing barriers to entry, and with an awareness of issues of practicality, we propose the possibility of a particular scenario that may yield reliable results. One NASA study (Zamani et al., 2019), examines utilizing computer vision (CV) algorithms over video recordings to detect and classify MMODs. With a lack of precise tools available to the average potential participant globally, it may be considered feasible, then, that a hobbyist with a video-capturing optical telescope (albeit, a high-end, professional-consumer unit), as well as any device on which to run CV algorithms for post-processing (such as the one in the aforementioned study), could aspire to contribute competitive data to the protocol. Such a class of contributor may be considerably likelier than an institutional source to add noise to the aggregated model, however we will discuss a solution to reduce and remove unsubstantiated data from that model in Section 2.3, as well as a method to maximize pursuit of accuracy in Section 3.2. Alternative CV techniques for the purpose of object detection are YOLO (You Only Look Once) or Faster R-CNN (Region-based Convolutional Neural Networks), among others.

Information gathered using these techniques could additionally be potentially integrated or cross-compared with other data sources (such as radar or satellite tracking data from institutional resources) to improve the accuracy of orbital element estimation. Additionally, machine learning algorithms could be employed to improve the system's accuracy over time by learning from labeled data sets of known orbital debris objects. We will discuss one such solution for prediction and interpolation in Section 2.4.

2.2 Synchronized Epochs

If Identifier dataset submissions were expected to happen freely and asynchronously, aggregation of those sets (discussed in Section 2.3) is likely impossible as reported points are improbable to match. We propose synchronizing submissions to a target frame in time, with a given parity between submission events, which we will denote these events each as epoch ϵ. This target time frame will be the start timestamp of the ϵ. Epochs, along with their target time frame for submission, will be known beforehand by all network participants. The responsibility of incrementing the epoch, as well as an illustration of the protocolization will be addressed in Section 3.2.

One might be concerned that even these data submissions will not match sufficiently, as most orbital debris circles the Earth at speeds of 7-8 kilometers per second [4]. This will inform our tolerance values when producing the data consensus model in Section 2.3.

The tradeoff for this solution is presented as a "framerate" issue; depending on the frequency of submission openings, there will be gaps where data is not being captured, incurring risk, whereas in the asynchronous model, we can posit that the distribution of submissions over time would naturally converge on a higher fidelity. This issue will be addressed in Section 2.4, where we will put together a collection of static images, submitted at incrementing values of ϵ, into motion and model future behavior.

Due to constraints regarding network cost basis and in the interest of making conservative projections for funding requirements, which both increase linearly in relation to epoch parity, we estimate a reasonable target for these epochs to be a few hours. If that parity seems too long or too short to the reader, it is worth noting that, in the early phases of this project, it is most likely that this protocol would merely be a supplemental, auxiliary option for existing centralized infrastructure tasked with this purpose, and the possibility of altering or improving this target as the network scales will be evaluated in Section 3.10.

2.3 Data Consensus

In the proposed protocol, this is what any given datapoint, representing an MMOD object in active orbit, produced by an Identifier will look like:

$$(x_i,\, y_i,\, z_i,\, s_i,\, C_i)$$

Where the subset $(x,\, y,\, z)$ represent Cartesian coordinates in space, relative to an origin at the center of the Earth $(0, 0, 0)$, s represents the speed scalar of the object at the time of observation, C represents a confidence score between 0.0 and 1.0, and i is the index of the point in their dataset.

The choice to represent datapoints unconventionally here arose from a compromise between fidelity, compute and storage cost, and network inclusivity. Traditionally, most observations performed by institutional sources take the form of TLE data, all of the information needed to reconstruct a target object's orbit. However, we will demonstrating a method to reconstruct and predict orbits using just this set of data in Section 2.3. Using simpler point data, we establish a lower barrier to entry for the ecosystem when it comes to onboarding new Identifiers, generating desirable distribution for the protocol. As for computational and storage rate, even a dataset composed of simple points such as these would produce a prohibitive cost basis for the network on most EVMs. While it is not unfeasible to imagine that future network scaling would yield opportunities to return to this decision and reconstruct the model such that we include Keplerian elements, for example, to identify orbits, we take this consideration further and outline a solution for data management in Section 3.2.

Each Identifier is tasked with collecting as many datapoints as they can per epoch ϵ. Since we are using synchronized epochs, they will be providing a dataset comprised of all points they are able to observe, as well as predict using algorithms such as the one outlined in Section 2.3, at ϵ. When the

epoch occurs, Identifiers will all submit their varying datasets with varying degrees of noise and confidence scores, and the protocol is tasked with aggregating this data into a refined image for ϵ.

The combination of Identifier datasets will happen as a three step process, consisting of labeling, aggregation, and thresholding:

1. Labeling. First, flatten the collection of datasets into a singular set P. Given the flattened set of points, the labeling process groups together points that are considered contiguous in regards to a given tolerance value ϖ. Let $K = \{l_1, l_2, \ldots, l_m\}$ denote the labeled points after the labeling process, where each labeled point l_j is represented by a new tuple $(x_j, y_j, z_j, s_j, C_j, label_j)$ and $label_j$ is an integer representing the label assigned to the group of points. The integer is raised with each new group formed.

The labeling process for each point is defined as follows:

a. For each point, p_i in P, we want to determine if it is sufficiently adjacent to any existing labeled point l_j. We can use the Euclidean distance formula to calculate the distance between two points' Cartesian coordinates (x_1, y_1, z_1) and (x_2, y_2, z_2). Recall:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

b. As noted in the parent item, we can use the distance threshold ϖ to determine if two points are sufficiently adjacent to be grouped together under the same label. If $d(p_i, l_j) \leq \varpi$ for any existing labeled point l_j, assign p_i the label of l_j

c. If p_i is not sufficiently adjacent to any existing labeled point, it is assigned a new label.

We then repeat this process for all points in P to obtain the set of labeled points L. The label assigned to each group of points allows for easy identification and aggregation of points belonging to the same group. Computational resource consumption for this process be categorized as $O(n^2)$.

2. Aggregating. Given the set of labeled points L, the aggregation process combines points with the same label, calculating the mean of their coordinates and a weighted average of their confidence scores based on the number of corroborations for each point.

Let $A = \{a_1, a_2, \ldots, a_k\}$ represent the aggregated points result, where each aggregated point a_i is represented by the tuple $(x_i, y_i, z_i, s_i, C_i)$.

The aggregation process is defined as follows. For each labeled group of points G_l:

a. Calculate the mean coordinates $(\overline{x_l}, \overline{y_l}, \overline{z_l})$ of the points in the group:

$$\overline{x}_l = \frac{1}{N_l}\sum_{j=1}^{N_l} x_j, \quad \overline{y}_l = \frac{1}{N_l}\sum_{j=1}^{N_l} y_j, \quad \overline{z}_l = \frac{1}{N_l}\sum_{j=1}^{N_l} z_j$$

Where N_l is the number of points in group G_l.

b. Calculate the weighted average confidence score $\overline{C_l}$ for the group, considering both the confidence scores of individual points and the number of points in the group. If there's only one

point in the group, we consider this to be an outlier and assign an aggregate confidence score of zero (0). First, we get the weighted sum of squared confidence scores:

$$sumC_l = \sum_{i=1}^{N_l} (C_i \cdot N_l)^2$$

Next, we normalize the weighted sum to ensure it stays between 0.0 and 1.0.

$$\overline{C}_l = \min\left(\max\left(0.0, \frac{\sqrt{sumC_l}}{N_l^2}\right), 1.0\right)$$

Where $\sqrt{sumC_l}$ is the square root of the weighted sum of squared confidence scores, N_l is the count of points in that group, $max(0, \circ)$ ensures that the result is not less than 0.0, and $min(\circ, 1.0)$ ensures that the result is not greater than 1.0.

3. Thresholding. After the aggregation process, remove points from the aggregated dataset whose confidence scores fall below a threshold value Ω. This results in the final dataset.

This aggregated data consensus image at a given epoch ϵ will be one of many images that ultimately compose a data consensus model in motion. Each image will be timestamp for ϵ at which the datapoints were all measured. See more on synchronized epochs in Section 3.2, and putting the model in motion in Section 2.3.

2.4 Prediction

The Keplerian Motion Model, formulated by Johannes Kepler in the early 17th century, describes the motion of objects in space under the influence of gravitational forces. As such, orbital debris also follow Keplerian motion due to the gravitational influence of the Earth. Predicting the location of these debris objects and describing their orbit is crucial to both producing a model that is useful to external parties with orbital interests, as well as increasing the viability of Identifier data production.

While the trajectory of undisturbed debris is most likely to follow the Keplerian Motion Model in orbit, trajectory-disrupting events do occur most often in the form of spacecraft launches, atmospheric drag, solar activity, and collisions between other approximately adjacent objects [3]. This increases the need for frequent, accurate data, although admittedly, there, again, exists a tradeoff between demands of higher fidelity and network operational cost basis. Since it is reasonable to expect a compromise will be made (see: Section 3.2*), and orbital data will not be shared in real time, we will require a method to extrapolate traditional classification data – namely these six orbital elements which describe orbital paths – from a series of images with a potentially wide distribution over time in order to yield data that's ultimately beneficial to external parties.

In order to carry out this process, which should demonstrate the capacity of this protocol to produce valuable data, we will be engaging in a two-step process to yield the best fit orbital ellipse that the points describe. This process could be described as a bit of a "chicken or egg" problem, with point association, the first step, challenged by a lack of elliptical data, which we produce in the second step, regression.

1. Association. The first step in creating a predictive model is similar to the labeling step in the data consensus process. We associate points across images, given high belief that they are the same object, in order to generate a descriptive flight path. Instead of a generic measurement likelihood, we

can model the likelihood of a measurement given a target state of an elliptical orbit.[1] Here we will be employing the Iterative Closest Point (ICP) algorithm to associate points across a series of timestamped images.

Given two point clouds P_j and P_{ref}, where each point is represented as p_i and ref_k respectively, our objective is to find the optimal transformation matrix T_i that aligns P_j with P_{ref} by minimizing the distance between corresponding points. The transformation matrix T_i describes the rotation and translation required to align one point cloud with another. It is typically represented as a 4x4 homogenous transformation matrix, where the rotation matrix R_i occupies the upper-left 3x3 submatrix, and the translation vector u_i occupies the rightmost column. The bottom row is typically [0 0 0 1] to maintain compatibility with homogenous coordinates.

$$T_i = \begin{bmatrix} R_{11} & R_{12} & R_{13} & u_1 \\ R_{21} & R_{22} & R_{23} & u_2 \\ R_{31} & R_{32} & R_{33} & u_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where R_{ij} are the elements of the 3x3 rotation submatrix R and u is a 3x1 translation vector.

Let p_{ij} represent the Cartesian coordinates of points in P_j, and recall that ref_k represents the Cartesian coordinates of points in P_{ref}. We aim to find an R_i and u_i that minimize the error metric:

$$E(R_i, u_i) = \sum_{i=1}^{N} \left\| R_i p_{ij} + u_i - ref_{k(j)} \right\|^2$$

where $k(j)$ denotes the index of the nearest neighbor of p_i in P_{ref}.

The transformation T_i that minimizes $E(R_i, u_i)$ can be estimated iteratively using methods such as least squares fitting or Singular Value Decomposition (SVD). We will be using SVD in our example.

We can also factor in a prediction based on the recorded speed scalar to compliment this method. Given a point p_{ij}, with speed s_{ij} in image Ψ, the expected position p_i in the reference image P_{ref} after a time interval Δt can be predicted using dynamic equations that account for potentially varying speed along the trajectory (objects in orbit will accelerate around the minor axis vertices), we first calculate the velocity vector v_{ij} for each point p_{ij} using the displacement vector Δr_{ij} for each point p_{ij} relative to its previous position and time interval Δt. Review:

$$v_{ij} = \frac{\Delta r_{ij}}{\Delta t}$$

We can modify the point cloud registration algorithm to incorporate the predicted expected positions p_{ig} as feedback.

Finally, we search for nearest neighbors. To associate points between P_j and P_{ref}, we search for the nearest neighbor ref_k in P_{ref} (via Euclidean) within a given tolerance distance Υ around the predicted position p_{ij}. If $\|ref_k - p_{ij}\| < \Upsilon$, then p_{ij} is associated with ref_k.

[1] Some risk is considerable with this operation. Trajectory-disrupting events can occur.

2. Regression. Least squares regression is a method used to fit a model to observed data by minimizing the sum of the squares of the differences between the observed and predicted values. In the context of estimating an orbital path from positional data, we can use least squares regression to find the best-fit ellipse, along with its Keplerian orbital elements, that minimizes the difference between the observed positions and the positions predicted by the orbit model.

The elliptical orbit model can be expressed using parametric equations, relating the coordinates of the object to several orbital parameters, enabling us to minimize for them:

$$x(t) = X_c + a \cdot (\cos(\theta) - e)$$
$$y(t) = Y_c + a \cdot \sqrt{1 - e^2} \cdot \sin(\theta)$$
$$z(t) = Z_c$$

Where:

X_c, Y_c, Z_c are the coordinates of the centerpoint of the ellipse,

a is the semi-major axis,

e is the eccentricity,

θ is the true anomaly, representing the angular position of the object in orbit.

We define the individual error function E_j as separate from L, the cumulative error function or loss function. E_j is defined as follows:

$$E_j = \left(\frac{C_g^\alpha}{Q\left[C_g^\alpha\right]} \right) \cdot \sqrt{\left(\frac{x(t_j) - x_j}{\delta x_j} \right)^2 + \left(\frac{y(t_j) - y_j}{\delta y_j} \right)^2 + \left(\frac{z(t_j) - z_j}{\delta z_j} \right)^2}$$

Where:

E_j is the weighted error for data point j,

$Q\left[C_j^{\alpha}\right]$ is the expected value of the confidence score modifier C_j^{α},

(x_j, y_j, z_j) are the observed coordinates,

$(x(t_j), y(t_j), z(t_j))$ are the predicted coordinates at time t_j.

$\delta x_j, \delta y_j, \delta z_j$ are the expected displacements based on estimated speed,

β is a coefficient that determines the importance of the eccentricity deviation term in the error function[2].

L is defined as follows:

$$L = \sum_{j=1}^{N} (E_j + \beta \cdot |e_j - e|)$$

Where:

N is the total number of observed data points,

e is the expected eccentricity of the orbit,

e_j is the estimated eccentricity of the fitted orbit at data point j.

[2] This is a term in the error function that penalizes deviations from the expected eccentricity, measuring difference between estimated and expected.

First, we make an initial guess for values a, e, θ, X_c, Y_c, and Z_c. Next, we employ an optimization algorithm, such as gradient descent or Levenberg-Marquardt algorithm, to minimize the loss function L and find the optimal values of a, e, and θ, as well as X_c, Y_c, and Z_c, although the latter set of values are likely to hover around the center of the Earth, which, as stated previously in Section 3.2, will be the origin $(0, 0, 0)$. This involves iteratively updating the parameters until convergence to a minimum of the loss function is achieved. Once the optimization process converges, it is essential to validate the fitted model against the observed data. This validation may involve visual inspection of the model's fit, statistical analysis of residuals, and comparison with independent data if available. If the fitted model does not adequately capture the observed data, further refinement may be necessary.

This technique could be used to interpolate real-time data for utilization by outside parties with orbital interests. It would likely have to be used by Identifiers to widen the range of potential captures, with a descending gradient of confidence scores.

3 Ptolemy

We begin the approach of demonstrating a proof of concept by introducing the target decentralized incentivization model in Section 3.1. There, we cover a number of problem statements which will be addressed in Sections 3.2-3.6. In Section 3.7, we synthesize, based on our aggregated solutions, the formula for calculating token reward share as a function of data production by Identifiers. In Section 3.8, we propose a method of handling redemption that minimizes cost basis for Identifiers. We show a tokenomics model for raising value for reward tokens in Section 3.9. Finally, we discuss a model for decentralized governance of the protocol in Section 3.10.

3.1 Ptolemy

Ethereum (ETH) [2] is a global peer-to-peer network of devices individually operating stack-based virtual machine software that executes transactions manages consensus over a ledger of transactions divided into blocks whose continuous geometry could be represented by an acyclic graph. Transactions are recorded and made transparent and available. They can individually be a basic exchange (sending) of the fee token, also called Ethereum (or ether), between public anonymous addresses[3], or they could be calls to endpoints in globally-deployed, addressed software applications, called smart contracts. Addresses, whether an agent or a contract, are represented as 40-byte hex which is a truncated public key in a traditional Diffie-Hellman (ECDH) asymmetrical public-private key pair elliptical curve scheme, where the private key is colloquially referred to as a signer (due to its usage to sign transactions to be produced by the public address and delivered on-chain), frequently represented as a mnemonic of 12 words called a seed phrase. Smart contracts, developed using a turing-complete software language (solidity), as well as data belonging to the contracts, is stored redundantly across all devices in the network (often in different measures depending on individual capacity). Whether or not transactions engage smart contracts or are simply an exchange of Ethereum tokens, fees are paid in those fee tokens. The fee is a bid for a transaction to be placed into the soonest "block" (nodes in the acyclic graph), delivered to the "block-builder", a device assigned by the consensus layer of the network to construct that next block in the ledger. Participants with addresses do not necessarily engage with the consensus layer, nor necessarily run the virtual machine software, but instead may deliver their transactions to the network often through usage of a "wallet", which may be local software nested in a user's internet browser (e.g. Metamask in Chrome), or an external piece of hardware (e.g. Ledger),

[3] Ethereum tokens can also be sent to smart contracts via this operation.

The EVM has been chosen as the target host for the decentralization component of this protocol due to its ability to host deployed, open-source, transparent, and distributed software, as well as its security model. It is worth noting that, in the context of realizing this project, with considerations toward economic, security, and behavioral tradeoffs, there may be multiple potential EVMs other than Ethereum to analyze as candidates to host the protocol.

Ethereum is not only a virtual machine network and a platform for conducting intelligent finance, but a protocol that might be considered a secondary or ternary layer of the internet. The protocol is mediated via governance (see Section 3.10), consensus, and public forums which host a line of Ethereum Request for Comments (ERCs). ERCs are reminiscent of the dawn of the internet we know today (RFCs). One of the earliest written ERCs is ERC-20, which puts forth a common standard for an exchangeable token smart contract.

Our ERC-20 token will be called Ptolemy (PTO). We will have a smart contract group, a decentralized application (Dapp), that manages functionality such as exchange, mint, burn, stake, slash, governance, data, and moderation. Ptolemy tokens are claimed in regulated portions to Identifiers as a reward for the successful delivery of considerably corroborated data. As an Identifier, your reward in Ptolemy tokens scales as more neighbors endorse the set of points you submit, and with increasing amounts of points, with diminishing returns per individual endorsement, per submission over time, and per point (thus, we can target an ideal minimum of neighbors, an ideal minimum of points, as well as an ideal frequency of submissions per participant).

Where does the value of the Ptolemy token come from? This issue will be addressed in Section 3.9. For now, let us proceed with the assumption that it can carry value via open exchange, and that said value will be dynamic.

3.2 Copernicus

The first problem of note with the concept of running such a protocol on Ethereum mainnet is that the cost of submitting point data directly to chain increases with more points delivered (although arguably more points potentially may yield more rewards, but in diminishing scale), and there is a limit to the size of the block as well.

However, as stated, this speculation may not actually matter; at potentially thousands of submitted points (depending on Identifier equipment grade and resources) being stored in the smart contracts (a costly operation), it would even be possible to hit the block's gas limit; 30 million gas units at the time of this writing [12]. It is simply no good to force Identifiers to pick only their favorite set of points, as this will corrupt our protocol's results. Even if there were an EVM network that could support the average calldata size, we should still expect the cost basis to be prohibitive, or at the very least, undesirable. Further reasoning for this can be found in the modeling for token value in Section 3.9.

The solution for this problem will involve moving some operations off-chain. Essentially, what is required is an off-chain cost-efficient peer-to-peer network to handle the distribution of the actual point data among Identifiers, while a compression (Merkle trees, Section 3.8) is recorded on-chain. The protocol for this peer-to-peer network could operate in an analogous fashion to the existing consensus layer used by the EVM to arbitrate the building of blocks. We will call this network layer and its client application Copernicus.

The concern here becomes, if points are all shared off-chain, how do we determine Identifier reward allocations on-chain?

In the space, there is an EVM architecture called a rollup. Rollups are aptly named, as they "roll up" bundled (via hashing or Merkle tree variants) transactions onto an existing network; typically one that might be considered costly but highly secure, such as Ethereum mainnet. One such sub-architecture for rollups is the optimistic rollup (most famously featured as of today by the EVM-equivalent rollup, Optimism). It is also aptly named; these architectures rely on a fraud-proof

scheme that assumes incoming, off-chain transactions are valid if they are submitted to chain and remain unchallenged for a sufficient period of time. Transactions might be challenged if they are not calculated correctly, resulting in a penalty for the rollup batch constructor [7].

Copernicus is an optimistic, application-specific, functionally-minimal, layer-2 rollup optimized for data submission, storage, and representation, as well as necessary computation used in the epoch lifecycle. It is intended to be a global registry cataloging and classifying LEO orbital debris. Here we will divide the design components and examine each:

1. Block Structure. Block time (parity between blocks) should be lengthy, block size should be maximized to ensure sufficient bandwidth for submissions. [TODO: Define the structure of blocks in the Copernicus blockchain. Each block should contain the necessary fields such as the block number, timestamp, hash of the previous block, epoch image, Merkle root, and other relevant metadata. Use ASCII art as needed.]

2. Data Storage Layer. Data storage will be partitioned into multiple sections:

a. Constants. Record of every constant value used in calculations for algorithmic opcodes and [TODO: Something goes here, like constants used in Copernicus network operation]. These values should be able to be modified via governance (Section 3.9). [TODO: This should be a hash table that just increments right?]

b. User Data. Will be a hash table indexed by 20-byte Ethereum addresses, value should have slots as follows:

```
| 64-bytes signature | 32-bytes staking | (MAX_POINTS*160)-bytes dataset |
```

The secp256k1 ECDSA signature is produced by the Identifier in the protocol epoch lifecycle, the staking is the amount the Identifier has staked, and the dataset is what the Identifier delivers, also in the epoch lifecycle. See Section 3.3 for more information.

c. Images. [TODO: Aggregated datasets per epoch that could be represented by an incrementing hash table (going up with each epoch)?] Will expand over time with each epoch, but could be a rolling window [TODO: Something here… also show the formula for this section's data size… if it's an incrementing hash table, then the values should be fixed-width array of maximum number of datapoints there can be in the aggregated data model (should be 30,000, LEO orbital debris count)]

[TODO: Anything else?]

… [TODO: How do we verify data integrity? The answer will obviously come in rollup form: we want to compress a representation of network data and deliver the compression (Merkle tree root) to Ethereum. Do we need group signatures here?]

3. Transaction Types. Transactional operations are limited. Copernicus is not Turing-complete. The operational flow of a transaction will consist of any necessary prefix opcodes, a single algorithmic opcode, and any necessary suffix opcodes. A potential list[4] of algorithmic opcodes available is presented:

```
0x01 || COMM || Called by Identifiers to commit a signature on their dataset.
```

[4] This list may be expanded in future iterations, as discussed in Section 3.10

```
0x02 || REVL || Called by Identifiers to submit a dataset to storage for an ongoing
     ||      || epoch. (Only available during the "reveal" phase.)
0x03 || .... ||
[TODO: Continue this list, adding as many opcodes as you can. They must be four
letters. Increment on left. READ for reading data, AGGR for data aggregation, TREE
for reward tree generation, and put it in a code box.]
```

No smart contracts are present. The native fee token used here should just be PTO, and a wrapper[5] ERC-20 on Ethereum mainnet for it should hold and indicate its value via decentralized exchanges present on Ethereum.

4. Networking and Communication. Copernicus consists of a network of nodes that perform various functions within the protocol. These nodes include:

a. Block Builders. Responsible for proposing and constructing new blocks containing aggregated data submissions for each epoch window.

b. Validators: Validate and verify the correctness of block submissions proposed by block builders, and, in general, monitoring the network for any anomalies or suspicious activities that may trigger challenges to block submissions that are deemed incorrect or fraudulent. Validators play a crucial role in maintaining the integrity of the network and ensuring consensus on the state of the blockchain.

c. Identifiers. Contribute observational data of LEO orbital debris to the network by submitting datasets for each epoch window. These data providers are essential participants in the protocol, as their submissions form the basis for generating the data consensus image.

... [TODO: Implement networking functionality to allow nodes to communicate with each other and synchronize the blockchain. This involves peer discovery, message propagation, and synchronization protocols.]

5. Resource Availability. [TODO: We should implement an RPC protocol for users (and other agents) to interface with the network, gather and view stored data.]

6. Block Proposal. [TODO: Block proposers (leaders) are chosen at random (using VRF) from a pool of nodes, weighted by PoS (or DPoS) as well as reputation (maybe ?)]

7. Bridge to Ethereum. [TODO: What needs to be "bridged" (and in which direction)? I think we need to bridge in both directions - for example, we'll have to bridge tokens

8. Proof of Stake. [TODO: Design and implement a staking and slashing mechanism to incentivize honest behavior and penalize malicious actors. This mechanism should involve staking tokens as collateral, detecting and penalizing fraudulent activities, and distributing rewards to honest participants. Basically just bridge tokens]

One problem of note in this design is that identifiers must pay network fees on Copernicus in PTO, and if the value of the reward tokens they receive falls below the cost of delivery of point data, the protocol's functionality fails. We will be exploring this problem further in Section 3.3.

[5] A wrapper contract refers to a token that represents an original token native to another network, and is preferably redeemable by means of bridging.

3.3 Lifecycle

As stated previously in Section 2.2, epochs, represented by ϵ, will mark the beginning of a submissions session or window. The parity between epochs will be indicated by τ. The number of blocks for the submission window will be referred to as Φ. We propose using the example number of blocks for Φ: 10. Additionally, we propose assigning epoch parity τ, at least initially, to be every hour, approximately 290 blocks apart. τ could be decreased as network funding scales. We will discuss modifying these variables in Section 3.10.

The protocol will begin at epoch $\epsilon = 0$, and increment the epoch counter with each passing instance, recording the current value in a smart contract within our on-chain application. Each epoch will require a degree of moderation. The responsibility of incrementing the epoch will fall upon moderators. [TODO: Define moderator role? Leave to governance?]

We will now be splitting a given epoch ϵ into three separate phases: first the "commit phase", then the "reveal phase", and finally the "distribution phase". Each phase will last a number of blocks. The commit phase lasts for block count α, the reveal phase lasts for block count β, and the delivery phase lasts for block count Γ.

Alice is an Identifier seeking to participate in the network. Tim, another network participant, is in a new role we will call the Aggregator. The process for that participation would be as follows:

1. Alice must stake[6] a number of Ptolemy tokens in a staking smart contract. The minimum amount that must be staked for Alice to receive rewards for participation will be referred to as δ. The amount that is staked can be represented by λ. When Alice later submits a transaction to receive her reward tokens, the reward allocation contract can check the staking contract to see if Alice's stake meets δ.

2. Tim, who is elected the role of Aggregator (see more on election in Section 3.10) for ϵ, must also be staked on-chain. The minimum stake required for the Aggregator role will be denoted as Ξ.

3. The commit phase of a new epoch ϵ occurs, and the target timestamp T for capturing points elapses. Alice, using her instrument's recordings as well as the Keplerian Motion Model (detailed in Section 2.2) for predicting the location of previously identified points, produces a set P of points. Points that are calculated using the prediction model, rather than being observed directly in the moment, will most likely have a descending gradient of confidence score C, which may vary based on how confident Alice is in her model's results as well as according to the visual range of her instruments.

4. Alice hashes (keccak256) an ABI-encoded tuple, (ϵ, N, P), where N is Alice's nonce[7], and the set P is Alice's point collection.

5. Alice signs (ECDSA) that hash using her signing key.[8]

6. Alice submits her signature to chain within the commit phase period α. The signature is stored in a mapping in a smart contract on-chain alongside that of other Identifiers participating in ϵ.

7. The reveal phase begins, and will last a number of blocks β. Alice is running the Copernicus client as an Identifier. The client broadcasts her set of points P, and begins to receive other point sets from other Identifiers (classified by their on-chain address) in return. The client accumulates these sets as Alice may want to aggregate the data for herself. Additionally, it would be likely the elected Aggregator would be known ahead of time, and their client endpoint may also then be known.

---

[6] Essentially, lock in a contract, in some cases like a rental deposit, in others like a savings account.
[7] A number used once, essentially a counter tied to each individual Identifier that will increment with each epoch they participate in.
[8] Practically speaking, she might do this using a hardware wallet or a browser extension but more ideally this could be an automated component of the Copernicus client software.

8. Tim is running the Copernicus client as an Aggregator. He is tasked with collecting the point sets of all Identifier participants for ϵ, including Alice, during the reveal phase. He may receive these point sets via normal broadcasting on the peer-to-peer network, or may receive them directly, as mentioned in (7).

9. Delivery phase begins. Tim has Γ blocks to produce a reward share Merkle tree. He takes the point sets received and produces a data consensus model using the process detailed in Section 2.3. He will then use the reward share formula, which we will determine in Section 3.7, to determine the score μ of each participating Identifier whose point sets he has received via Copernicus. This score μ should then be converted into a percentage share value for each Identifier.

10. Tim produces the reward share Merkle tree using all the calculated scores. He hashes (keccak256) the data consensus model, then signs it. He delivers the reward share Merkle tree root, along with the data consensus model signature, to chain within the Γ blocks.

11. The epoch ϵ ends. Alice can, at her convenience, submit a Merkle tree proof on-chain to claim her reward share of Ptolemy from ϵ's pool. She could also forego this process and thus "donate" her efforts to the cause. The window to submit the proof before expiry could be something like a week or a month.

Once points have been broadcasted, if a signature committed by a given Identifier in the commit phase does not match the point set they broadcasted, anyone can slash that Identifier's stake.

On the Copernicus peer-to-peer network, the commit phase begins as soon as T occurs. Alice is running the Copernicus client as an Identifier. The client broadcasts her signature on this network, and receives others' signatures in return. The client accumulates these signatures. No one is able to use these signatures yet to decode point data.

3.5 Pioneer Partial Oracles

[TODO]

3.6 The Crow Problem

[TODO]

3.7 Reward Share

The reward allocation accumulates as these corroborations occur, with the allocation being calculated using the formula:

Let:

p_i be a point in the individual subset.
$\overline{p_j}$ be an aggregated point from the aggregate model.
n is the number of points in the participant's dataset.
n_{max} is the maximum number of points before diminishing returns kick in.
$confidence(p_i)$ represent the confidence score of point p_i.
$distance(p_i, \overline{p_j})$ represent the Euclidean distance between p_i and p_j.
$penalty(p_i)$ represent the penalty for a point being too far away from other points.
$stake(s)$ represent the staking modifier, where s is the amount staked.
$count(n)$ represent the number of points in the Identifier's dataset.

Given these definitions, we can construct the scoring formula for each individual point as follows:

$$score(p_i) = confidence(p_i) \cdot \left(1 - \frac{distance(p_i, p_j)}{d_{max}}\right) \cdot \left(1 - \frac{penalty(p_i)}{p_{max}}\right) \cdot \left(1 + \frac{stake(s)}{s_{max}}\right) \cdot \left(1 + \frac{count(n)}{n_{max}}\right)$$

Where:

d_{max} is the maximum allowed distance before a penalty is incurred.
p_{max} is the maximum allowable penalty.
s_{max} is the maximum staking amount before diminishing returns kick in.

To ensure diminishing returns, we can use a function like the square root or logarithm of the staking amount. For example:

$$stake(s) = \sqrt{s}$$
OR
$$stake(s) = log(s + 1)$$

These functions would ensure that increasing staking amounts yield diminishing returns in scoring. We can adjust the parameters of these functions to fine-tune the level of diminishing returns. See more about this in Section 3.10, where we discuss governance.

Additionally, we can use either of the same two functions to achieve diminishing returns for the number of points that the Identifier submits. By incorporating this additional modifier into the scoring formula, we ensure that Identifiers are rewarded for contributing more points to the dataset while preventing excessive rewards for large datasets. Note that we will discuss the actual maximum number of points that an Identifier can submit, again, in Section 3.10.

$$count(n) = \sqrt{n}$$
OR
$$count(n) = log(n + 1)$$

Similarly, we can design the penalty function to ensure that penalties increase with decreasing confidence and increasing distance. For example:

$$penalty(p_i) = \Lambda \cdot (1 - confidence(p_i)) \cdot distance\left(p_i, \overline{p}_j\right)$$

Where Λ is a scaling factor to adjust the penalty amount.

By combining these components, we can create a scoring formula that incorporates the confidence of the points, their distance from the aggregate model, penalties for outliers, and the staking amounts, while ensuring diminishing returns for staking, penalties, and count.

3.8 One-Way Burn Pools

[TODO]

3.9 Protocol Governance

[TODO]

2
https://www.nasa.gov/centers-and-facilities/nesc/space-debris-understanding-the-risks-to-nasa-space craft/
3 https://ntrs.nasa.gov/api/citations/20190001193/downloads/20190001193.pdf
4 https://sma.nasa.gov/sma-disciplines/orbital-debris

7 https://ethereum.org/en/developers/docs/scaling/optimistic-rollups/

12 https://ethereum.org/en/developers/docs/blocks/

18 https://ntrs.nasa.gov/api/citations/20190032383/downloads/20190032383.pdf
https://github.com/nnategh/AMOS2019

13 https://en.wikipedia.org/wiki/Space_debris