# Toward a Marketplace for Aerial Computing

Arjun Balasingam* Karthik Gopalakrishnan* Radhika Mittal†
Mohammad Alizadeh* Hamsa Balakrishnan* Hari Balakrishnan*
*Massachusetts Institute of Technology †University of Illinois at Urbana-Champaign

## ABSTRACT

The rapid proliferation of commodity drones has expanded interest in building applications that acquire imagery, video, and sensor data at scale. In addition, recent work on drone programming frameworks have simplified the development of aerial computing apps that gather this data. These advancements have popularized the drones-as-a-service model, where large drone fleets serve multiple apps simultaneously.

This paper proposes a marketplace for aerial computing, where apps can gather aerial data on demand and providers can offer up their drones for aerial computing. We introduce Aerialis, a drones-as-a-service platform that schedules tasks to drones by arbitrating bids submitted by apps. Aerialis allows apps with different semantics and spatiotemporal preferences to express how much they would like to pay for each aerial computing task. It then aggregates requests across apps, and schedules tasks on drones according to a marketplace policy (e.g., maximizing revenue or guaranteeing quality-of-service to apps). We build a prototype of Aerialis, and implement urban sensing apps to monitor air pollution, measure road traffic, and profile cellular throughput. We discuss operational challenges in deploying Aerialis, and show how the measurements collected from our real-world experiments offer valuable insights for engineers and city planners.

## 1 Introduction

Consumer drones today come equipped with a variety of commodity sensors and standard communication capabilities [1, 2]. The versatility of these drones has generated an increased interest in the development of *aerial computing* applications [6, 8, 12, 15], which gather and analyze large amounts of data, such as imagery, video, air quality, etc. Additionally, recent work has introduced general-purpose programming languages [7, 9], which are aimed at supporting the variety of complexities and constraints required by these sensing apps.

Emerging from this excitement around aerial computing is the *drones-as-a-service* model, where apps are decoupled from drone infrastructure. In these platforms, developers submit apps (which specify tasks, e.g., measure air quality, deliver a package, record video, etc.) to a platform that then schedules these tasks on a fleet of drones.

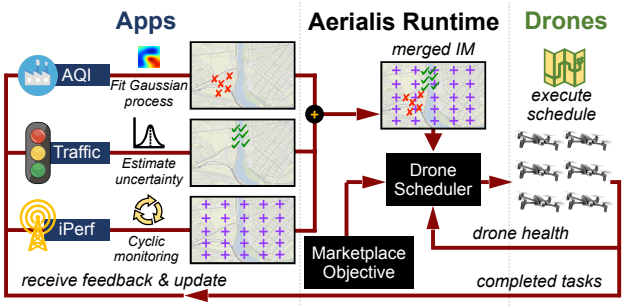


Figure 1: Aerialis is an aerial computing marketplace that allows apps to complete sensing and data acquisition tasks on demand.

A key benefit of a shared drone platform is the ability to multiplex apps with co-located tasks on the same fleet of drones, allowing operators to amortize flight (energy) and hardware costs. For example, a fleet of city-owned drones can simultaneously deliver packages, gather data about air pollution [3], monitor parking spots [4], measure road traffic, and identify dangerous incidents. Such multiplexing is more efficient than each app using its own drones, because an individual drone can reduce wasteful long flights and focus on sensing data for multiple apps in nearby areas.

We envision a future where a shared economy of drones can facilitate a *marketplace for aerial computing*. Much like cloud computing (e.g., Amazon EC2, Google Cloud, Microsoft Azure), app developers can acquire aerial data on demand, while drone operators can profit by offering any idle time on their drones to support these computing apps. However, to realize this vision, we need a marketplace arbiter that can (i) ingest and synthesize requests from a variety of apps and (ii) seamlessly coordinate a drone fleet with heterogeneous sensing and computing capabilities. This is challenging because apps could have a variety of spatiotemporal requirements, and drone time is a scarce resource.

In this paper, we introduce *Aerialis*, a drones-as-a-service platform that facilitates a marketplace for aerial computing. Aerialis's design is centered on the interest map, a narrow-waist abstraction that interfaces apps with a drone scheduler. Through this interface, an app expresses how much it would like to pay for each aerial computing task. Interest maps allow apps to (i) specify atomic tasks, and (ii) encode relative preferences amongst tasks, so the sched-

uler knows what to prioritize when there is not a sufficient number of drones to fulfill all tasks. After receiving interest maps from all subscribing apps, Aerialis aggregates requests and preferences across apps and computes a schedule for each drone according to a marketplace objective (e.g., maximize total revenue, balance drone workload, or guarantee a quality-of-service to apps). Aerialis replans drone routes in response to updates in interest maps, drone availability, and travel time uncertainty. Fig. 1 shows an overview of Aerialis.

In §2, we use an example to further illustrate the scheduling challenges that motivate the need for a modular and expressive platform like Aerialis. §3 presents our design and implementation of Aerialis, and highlights operational challenges in building a robust system. Then, in §4, we describe our experience implementing three real-world urban sensing apps atop Aerialis. Finally, in §5, we discuss how the framework Aerialis proposes can be extended to support more marketplace objectives and incentive-compatible pricing schemes.

## 2 Challenges and Requirements

As a marketplace arbiter, Aerialis must match drone resources to app requests, while maximizing drone utilization and exposing a flexible interface to apps. This introduces several new challenges.

| App | Requirements |
|---|---|
| deliver packages | • streaming requests<br>• some packages are prioritized |
| patrol traffic | • patrolling requires continuous tracking<br>• tracking logic should be private |
| map street parking | • detect/sample high-traffic areas more<br>• maintain fresh measurements |

Table 1: Aerial computing apps have diverse requirements.

**Challenge #1: App semantics and objectives.** Table 1 lists some requirements of three different aerial computing apps that may run atop a shared drone computing platform. Notice that each app has unique semantics: while the package delivery app may request tasks from several discrete locations, the traffic patrol app would want a drone to track a vehicle along a waypath for a period of time. At the same time, the traffic patrol app may not want to expose its tracking code (e.g., model-based prediction) to the platform. Additionally, since the platform may not be able to immediately service all tasks due to resource constraints, the package delivery app may want to express a relative ordering amongst tasks.

**Challenge #2: Resource constraints.** Commodity drones typically have 30-minute flight times (on a single charge), and require an expensive flight back home to recharge. Further, there are often far more tasks than can be completed in this duration. A shared platform should seek to minimize wasted flight time, in addition to optimizing the marketplace objective (e.g., revenue or quality-of-service).

**Challenge #3: Environment uncertainty.** Drones could fail mid-flight, and the platform should adapt and continue to service subscribed apps. Additionally, apps could be volatile and change their sensing preferences dynamically. A shared aerial computing platform must be robust to dynamic environments.

Aerialis addresses these challenges by exposing a narrow-waist, expressive interface for apps. Each app can prioritize its tasks, and add, cancel, and update tasks as needed. Aerialis's abstraction allows apps to participate in an aerial computing marketplace by placing bids to acquire sensor data or complete a mission at specific locations. Aerialis then optimizes all incoming bids, applying a marketplace objective, to allocate tasks to drones.

**Related work.** Aerialis is motivated by the popularity of acquiring aerial data on demand. Prior works propose programming primitives to write aerial sensing apps and study onboard security for shared drone computing; however, they do not address the scheduling challenges that arise when multiple apps coexist in a marketplace. For instance, AnDrone [14], a drones-as-a-service platform, provides a framework to share onboard drone computing resources (i.e., CPU, camera, sensors, etc.) in a manner that preserves privacy amongst apps that the same drone simultaneously services. Voltron [9] is a general-purpose programming interface for reactive sensing apps; however it assumes that apps have exclusive access to a drone fleet, and thus does not allow apps the flexibility of expressing relative preferences amongst tasks, which is valuable when apps contend for resources in a shared platform. BeeCluster [7] predicts app demand in order to boost the efficiency of a drone fleet. We believe this work on security and programming interfaces is complementary to the marketplace architecture proposed by Aerialis.

## 3 Design of Aerialis

Aerialis exposes an abstraction, called an interest map, through which apps can express and update their desired tasks in the form of bids (e.g., a dollar value per task). As depicted in Fig. 1, at runtime, Aerialis queries apps for interest maps, computes a schedule for each drone, and charges apps according to a pricing policy. Aerialis computes schedules for a fixed horizon (e.g., every 15-minute round-trip flight). However, it replans more frequently (e.g., every 5 minutes) in order to (i) allow apps to update their interest maps based on changes in preferences and (ii) respond to changes in drone availability and travel time uncertainty.

### 3.1 Interest Maps

Aerialis interfaces apps with its scheduler via an abstraction called an interest map. An interest map is a set of tasks that an app would like the drone fleet to complete. At runtime, apps submit their respective interest maps to Aerialis, which merges them prior to allocating tasks to drones.

**Attributes.** An interest map is a set of tasks that an app would like to complete. Table 2 lists the attributes of each task. A `location` could be the GPS coordinate for an air quality or traffic measurement, or the start location for a video along a waypath. The `task` attribute specifies

| Attribute | Definition |
|---|---|
| `location` | GPS coordinate (i.e., lat, lon, altitude) |
| `task` | executable (e.g., Python code) to run on drone |
| `duration` | estimated duration for task |
| `interest` | bid (e.g., in dollars) for task |

Table 2: Attributes of an interest map entry.

any code to execute on the drone (e.g., collect PM2.5 measurement, record/analyze video, or take control of the drone) once the drone reaches the desired `location`. `duration` states the estimated time to execute a task. Finally, each task also has an associated `interest`, which corresponds to the app's dollar-value for that particular task.[1] This attribute allows an app to encode a relative preference between tasks, which Aerialis uses to prioritize tasks when the platform is resource-constrained.

**Expressiveness.** Interest maps are suitable abstractions for the apps described in Table 1. For instance, each request in the package delivery app corresponds to an interest map entry. Each interest map entry in the traffic patrol app would specify a waypath (i.e., a *sequence* of locations to visit in order to complete the tracking task); Aerialis's scheduler only requires the start/end location and the (estimated) duration of the task. Package delivery apps could specify additional attributes like delivery time windows. The street parking app may use complex models to estimate traffic uncertainty; however, since Aerialis supports frequent replanning, this app could simply recompute new interest maps as needed. §4.2 describes our implementation of three urban sensing apps, and further highlights the versatility of interest maps.

**Merging interest maps.** An interest map is a powerful abstraction for multiplexing apps because it is lightweight and composable. In order to multiplex apps, Aerialis merges interest maps, by simply combining the sets of interest map entries submitted by all apps (see Fig. 1). It combines coincident task entries (i.e., identical locations) into a super task interest map entry, where the `interest` attribute is the sum of the bids specified by the individual tasks; drones execute tasks at such locations in parallel (unless multiple tasks require the same sensor). In composing the merged interest map, Aerialis also applies heuristics, such as clustering nearby tasks and collapsing them into a single location (e.g., centroid of the cluster), where the radius of the cluster is within the error tolerance of the subscribing apps. This pre-processing step helps boost the efficiency of the platform: we observed over many flights and weather conditions that decelerating to, hovering at, and accelerating from each distinct location drains the battery more rapidly than cruising at a constant speed.

### 3.2 Implementation

Fig. 1 provides an overview on an Aerialis deployment. Aerialis's runtime gathers interest maps from its apps, computes a schedule, and dispatches the drones. We implement

[1]Computing monetary value for a task is non-trivial; we explore some options in §4.2 and §5.

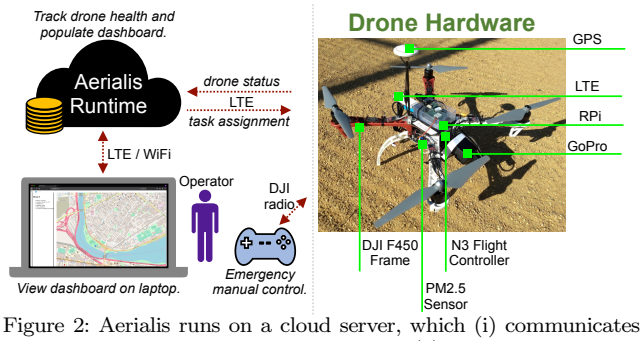


Figure 2: Aerialis runs on a cloud server, which (i) communicates with DJI F450 drones via a cellular link and (ii) updates drone health and app progress to a web dashboard.

Aerialis in Go; our software consists of three components, shown in Fig. 2: (i) Aerialis runtime, which is deployed on a cloud server, aggregates app requests, and computes a schedule; (ii) a dashboard exposing flight statistics, drone health, and app progress for the platform operator; and (iii) an onboard software stack that interfaces with the drone's flight controller and sensors.

**Drone hardware.** Our drones use the DJI F450 frame [2]; we customize the onboard electronics to support more sensors and programming flexibility. We mount a Raspberry Pi as our onboard computer, a PM2.5 sensor to gather air quality measurements, a GoPro to collect aerial videos, and an LTE dongle to communicate with the Aerialis runtime process. We use the DJI N3 flight controller, which comes with a C library for low-level flight control. Aerialis automates drone orchestration by issuing commands to the flight controller. However, we implement two failsafes to intervene in emergency situations:

- The drone regularly pings an iPerf server to probe the available cellular bandwidth. When the measured throughput is below 1 Mbps, the flight controller disengages from Aerialis and allows the operator to regain manual control.
- When the drone battery is low (i.e., $<10\%$ remaining), the flight controller disengages from Aerialis, and automatically navigates the drone back to its takeoff site.

Whenever a drone "leaves", Aerialis simply adapts its schedule to use the remaining drones. Aerialis's dashboard describes the real-time status of each drone (Fig. 2), so an operator can take over when a drone's flight controller triggers manual control.

**Aerialis runtime.** Aerialis relies on a centralized scheduling framework that orchestrates the drone fleet in unison. It aggregates the interest maps submitted by each app and computes a schedule according to its marketplace policy (examples described in §4.2). Aerialis computes schedules for a fixed time horizon based on resource constraints (e.g., one 15-minute round-trip flight) and specifies an ordered list of waypoints (and tasks) for each drone to complete. To compute these schedules, Aerialis uses standard vehicle routing solvers that maximize a weighted sum of fulfilled tasks.

We deploy the Aerialis runtime module (Fig. 1) on an

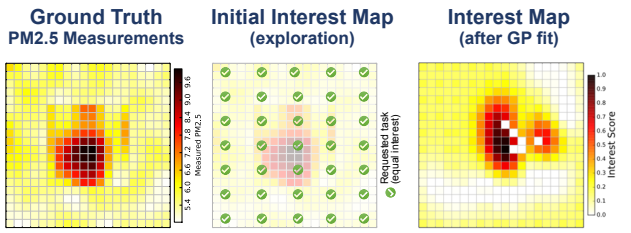


Figure 3: Snapshots from our implementation of an air quality mapping applications in Aerialis. This app uses a Gaussian Process to navigate the tradeoff between exploring the region of interest and collecting more useful measurements in the vicinity of the plume.

Amazon EC2 `t2.micro` instance. At boot time, each drone automatically establishes an HTTP connection with this server (via LTE), which allows (i) the server to notify the drone of its next assigned task and (ii) the drone to update the server of its health (i.e., battery status, location, current task) every second. Additionally, the server hosts a web dashboard to allow an operator to monitor drone health and app progress; Fig. 2 shows a snapshot from this dashboard.

**Aerial computing apps.** As described in §3.1, sensing apps interface with Aerialis via interest maps. App developers can customize their implementations to leverage complex modeling techniques (examples in §4.1) that interpret any gathered and decide on the next set of tasks to request of Aerialis. Apps run as standalone processes and simply post interest map updates to the Aerialis runtime process.

## 4 Urban Sensing Marketplace with Aerialis

We evaluate Aerialis on three urban sensing apps deployed in Cambridge, MA. Fig. 1 overlays the approximate sensing locations for each app on a map, and Table 3 summarizes their characteristics. In §4.1, we describe how we implemented each app atop Aerialis, and share some insights from data we gathered on the field. §4.2 quantifies the performance of Aerialis under different marketplace objectives. We tested our implementation of Aerialis using a fleet of 2 drones. However, to systematically compare app performance under different marketplace policies (§4.2), we also collect ground-truth traces for each app and evaluate with trace-driven emulation.

| App | # of Tasks | Task time | Update Logic |
|---|---|---|---|
| AQI | 40 | 20 sec | with time |
| Traffic | 11 | 30 sec | with time and meas. |
| iPerf | 100 | 10 sec | every meas. cycle |

Table 3: Characteristics of our three urban sensing apps.

### 4.1 Sensing Apps

**Monitoring air quality.** The *AQI App* seeks to estimate the geographical dispersion of a smoke plume quickly. However, since resource constraints are stringent and the app pays for each measurement, a simple grid search over the entire domain of interest would be expensive and inefficient. Atmospheric chemists approximate the dispersion of a plume using a Gaussian Plume Model [13]. Our app applies this model by using a Gaussian Process (GP) [10] with a Radial-Basis Function kernel; the GP estimates the plume PM2.5 concentration at each query location as a Gaussian distribution $G \sim (\mu, \sigma)$, with mean $\mu$ and standard deviation $\sigma$. Fig. 3 shows the PM2.5 concentration of a real plume that we measured near a freeway ramp during rush hour.

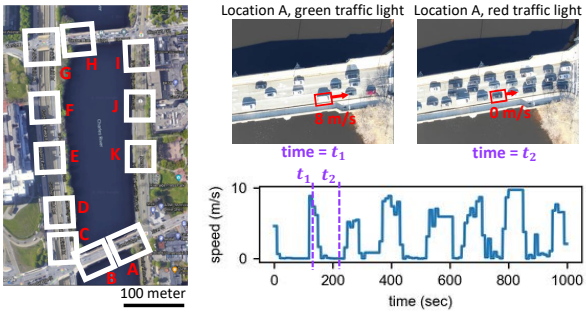


Figure 4: We collected traffic videos using drones at 11 locations labelled *A*-*K* (left). Our car detection and tracking algorithm computes average speeds, which show cyclic patterns near intersections depending on the state of the traffic light.

We derive relative preferences amongst sensing locations from the output of a GP model, and encode them via interest maps. Upon initialization, the app submits an interest map with sensing locations spaced out evenly over a grid (see "Initial Interest Map" in Fig. 3). To navigate the exploration-exploitation tradeoff, we model this search as a multi-armed bandit problem. For each potential measurement location, we set the `interest` proportional to $\mu + 2\sigma$; this allows the app to express that it would like to favor exploration (i.e., greater preference on more uncertain locations), but still be guided in the direction of high expected PM2.5 concentration (i.e., preference toward high $\mu$)[2]. Initially, $\sigma$ is high everywhere, and the app prefers to explore. Fig. 3 shows the GP fit after several initial samples are gathered; notice that the interest starts to drift toward the fringes of the plume. Aerialis thus provides a simple interface for sensing apps to leverage a complex model (privately), while still allowing apps to express fine-grained preferences amongst tasks.

**Sensing road traffic.** The *Traffic App* measures traffic congestion using aerial videos from 11 sites in a neighborhood, as shown in the map in Fig. 4. It estimates the average vehicular speeds through a simple video analytics pipeline, where it (i) identifies cars using a YOLO object detector [11], (ii) tracks each car using a Kernel Correlation Filter, and (iii) correlates identities of cars across frames based on a nearest-neighbor heuristic. Fig. 4 shows examples of the trajectories computed by the app. Aerial traffic monitoring offers the potential to obtain accurate, real-time data at a finer granularity (e.g., lane-level speed, blocked bike path, etc.) than possible with conventional GPS-trace based approaches. For example, in our experiments, we observed that when the traffic light at the edge of location *A* is red, cars tend to build up over the entire bridge

[2]This is known as the Upper Confidence Bound in the multi-armed bandit problem.

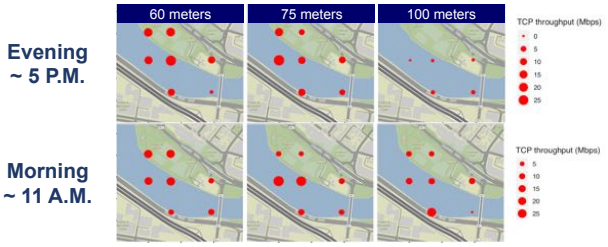


Figure 5: We implemented an app to profile cellular throughput in an urban area, in order to determine the viability of cellular communications to coordinate a shared fleet of drones.

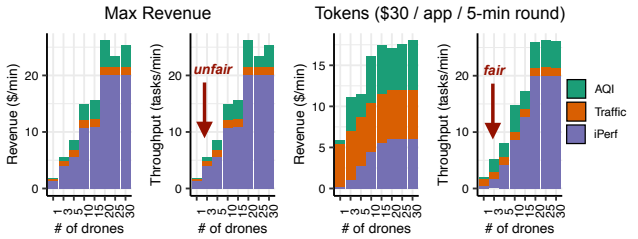


Figure 6: Aerialis can enforce different marketplace policies to control an app's contribution to the marketplace revenue.

(as confirmed by our measurements at location $B$), and spill over to the next traffic light. Such observations offer valuable insights to city planners and traffic engineers.

If sufficient drones are available (and the app developer was willing to pay for continuous drone time), Aerialis can schedule a drone over each of the desired locations. However, in practice, the app may want to prioritize certain measurement locations. Two competing factors govern this choice: (i) drones should prioritize and visit locations where the measurements are stale (i.e., last measured 10 minutes ago), and (ii) drones should visit locations which have shown high uncertainty $\sigma$ in speeds (e.g., near intersections). Thus, the app uses the following rule to compute the `interest` $I_l$ at location $l$:

$$I_l \propto \begin{cases} 10^2, \text{ if } \leq 3 \text{ meas. at } l \text{ in last 10 min} \\ \sigma[\text{Meas. at } l \text{ in last 10 min}], \text{ otherwise} \end{cases}$$

The app then recomputes an interest map whenever there is a new measurement, or when old measurements time out.

**Profiling cellular throughput.** The *iPerf App* aims to profile cellular throughput over an urban area; this data is valuable for autonomous drone operations that require robust aerial 5G communications. Fig. 5 shows some highlights from our measurements on the field, gathered near the Charles River at different times of day and at different altitudes. Interestingly, we found that average bandwidth was low ($< 5$ Mbps) during the 5 P.M. rush hour, likely because more cars are present on the roads. We also observed that, at altitudes of 60-75 meters, cellular throughput was sufficient for practical use cases; however, availability became more spotty at 100 meters.

While the iPerf app is not time-sensitive, it would like measurements from significantly more locations than the Traffic or AQI apps. Since the iPerf app is indifferent to location, it submits an interest map with identical `interest` values at all locations. In order to build a profile of bandwidth over time, this app resubmits the interest map after completing each cycle of 100 iPerf tasks.

### 4.2 Marketplace Policies

As described in §3, the interest map allows Aerialis to enforce different marketplace policies. In this section, we show how Aerialis schedules the three urban sensing apps described in §4.1 under two policies. We configure Aerialis to re-compute schedules every 5 minutes (so that it is sufficiently reactive to the apps), while returning to its takeoff location every 15 minutes to recharge. We also simulate drone fleets of different sizes. We consider two evaluation metrics: (i) revenue generated by Aerialis (in dollars earned per min) and (ii) throughput (in tasks completed per min). Fig. 6 summarizes our results.

**Maximizing revenue.** First, we consider a marketplace that maximizes total revenue. We assume for simplicity that each app places a baseline bid of \$1 for each sensing task, with slight variations—proportional to their `interest` update rules—to capture sensing preferences. Fig. 6 shows Aerialis consistently favors the iPerf app, since each iPerf measurement is easy to gather (i.e., 10 seconds/task). By contrast, the Traffic app only has 11 tasks, and each task is more expensive to complete, so Aerialis does not prioritize it and instead fulfills its tasks when it completes nearby iPerf tasks. In this example, we assume that all apps value each task at roughly \$1; however, if one app places a significantly higher bid, Aerialis would prioritize it to maximize revenue.

**Issuing tokens.** To provide apps a more equitable share of drone resources, we can allocate tokens (artificial currency) to apps, in order to constrain each app's bidding budget. This helps mitigate the effects of a "richer" app having a monopoly on drone resources. Fig. 6 shows the marketplace throughput and revenue when each app is allotted a pool of tokens worth \$30 in every 5-minute round. We find that when Aerialis enforces this policy, the Traffic and AQI apps begin to occupy larger shares of both marketplace throughput and revenue. Since the Traffic app only has at most 11 sensing tasks at any given time, each of its tasks has a higher value than a single iPerf task.

Note that this token scheme does not necessarily prioritize apps with fewer tasks; instead, it gives the same "purchasing power" to all apps. So, an app with more tasks (e.g., iPerf) can choose to put its tokens on a smaller subset of high-priority tasks in order to be competitive in obtaining drone time. Additionally, unused tokens can roll over to future scheduling intervals.

**Expanding drone fleet.** The benefits provided by interest maps and Aerialis's support for different marketplace policies are most evident when the shared drone platform is resource-constrained (i.e., more tasks than drones can fulfill). Fig. 6 indicates that both revenue and throughput saturate as the drone fleet gets larger (starting around 20 drones), because the platform is no longer resource-constrained.

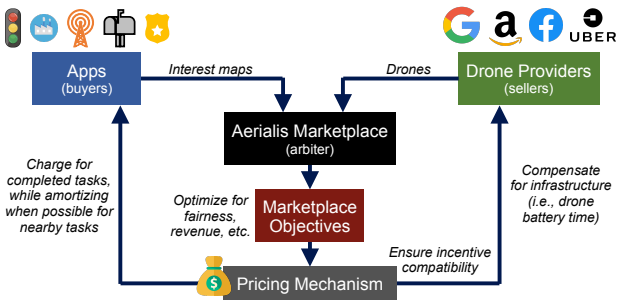


Figure 7: Aerialis can be extended to build out an aerial computing marketplace, with new platform objectives and pricing mechanisms.

## 5 Discussion

Fig. 7 lays out how Aerialis could be extended to build out a marketplace for aerial computing. In this section, we elaborate on some of our limiting assumptions and outline some ongoing and future work.

**Marketplace objectives.** Our evaluation (§4.2) covers scenarios where the overall objective was to maximize the revenue collected. However, as a marketplace arbiter, Aerialis may also want to satisfy other objectives. For instance, Mobius [5] proposes a scheduling algorithm for mobility platforms with the objective of providing provably good fairness over time. We are interested in supporting Mobius and other marketplace objectives, such as load balancing across vehicles and app service-level agreements.

**Value estimation.** This paper shows how apps can express relative preferences amongst their tasks, but assumes that apps know how to place an absolute *monetary value* on each task. However, just like the value of popular services such as cloud computing and ad auctions were only discovered with time, we believe that, as an aerial data marketplace matures, apps will be able to better estimate the value of a task.

**Truthful bidding.** While we assume for this paper that apps using Aerialis bid truthfully, in practice, they may be strategic in reporting their preferences. For example, after discovering that most drones are far away from a desired location, an app could first place a very high bid (e.g., $1000) near the region of interest, forcing at least one drone to move toward that location (assuming a revenue-maximizing scheduler). Then, just before that task is serviced, it could lower its bid on that task (e.g., 1¢), and the scheduler will still likely fulfill the task for a much lower price. A simple fix to this problem is to charge an app for a task once it is added to a schedule, instead of when the task gets fulfilled. However, this may hurt truthful apps that are *volatile* and cancel/change requests after gaining initial knowledge about the environment (e.g., traffic sensing or air pollution apps in §4.1). Thus, Aerialis would benefit from a pricing mechanism that incentivizes truthful bidding.

**Pricing schemes.** Currently, Aerialis simply charges each app the exact amount it bids. As a marketplace arbiter, Aerialis should also be able discount its price to account for the benefit from spatial multiplexing of nearby tasks from different apps (i.e., only charge for the amortized flight time). Future work includes developing heuristics to compute amortized cost. This is nontrivial because traveling 6 minutes to app $A$ and then 3 minutes to app $B$ does not mean that app $A$ should be charged 2× more than app $B$, since app $A$ is responsible for the short travel to app $B$.

**Security.** Data privacy is a concern in any drones-as-a-service deployment. AnDrone [14] provides a method to containerize onboard sensor logic, but does not conceal the location of the drone. However, in this marketplace setting, malicious apps could flood the system with thousands of cheap (e.g., 1¢) tasks spread everywhere in order to draw inferences about the spatiotemporal demand patterns and bidding strategies of other apps using the platform. We would like to extend Aerialis to be robust to these settings.

**Scope of interest maps.** Interest maps cannot encode continuous monitoring tasks or support combinatorial constraints (e.g., complete any 2 tasks from a set of 4 tasks). We would like to leverage complementary work from general-purpose drone programming frameworks [7,9] to expose a broader set of primitives to apps.

## 6 Conclusion

This paper proposed Aerialis, an aerial computing marketplace, where apps can complete aerial tasks on demand and providers offer up their drone computing services. We characterized the diverse spatiotemporal requirements of aerial computing apps, and identified the resulting scheduling challenges when a marketplace multiplexes several apps on the same fleet of drones. We implemented three real urban sensing apps atop Aerialis, demonstrated that it can support different marketplace policies, and proposed methods improve the flexibility and robustness of Aerialis. We hope to generalize the ideas that underpin Aerialis to build out a marketplace for acquiring data using mobility platforms.

## 7 References


[1] https://www.parrot.com/us/drones.
[2] https://www.dji.com/.
[3] https://www.ee.ucla.edu/mapping-air-pollution-from-a-drone/.
[4] http://datafromsky.com/news/smart-parking-using-drones/.
[5] A. Balasingam, K. Gopalakrishnan, R. Mittal, V. Arun, A. Saeed, M. Alizadeh, H. Balakrishnan, and H. Balakrishnan. Throughput-fairness tradeoffs in mobility platforms. https://www.dropbox.com/s/w3uzfzbiwkmilku/mobius-tr.pdf, 2021.
[6] A. Dhekne, A. Chakraborty, K. Sundaresan, and S. Rangarajan. Trackio: Tracking first responders inside-out. In *16th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '19. USENIX Association, 2019.
[7] S. He, F. Bastani, A. Balasingam, K. Gopalakrishnan, Z. Jiang, M. Alizadeh, H. Balakrishnan, M. J. Cafarella, T. Kraska, and S. Madden. Beecluster: drone orchestration via predictive optimization. In *MobiSys*, pages 299–311, 2020.
[8] W. Mao, Z. Zhang, L. Qiu, J. He, Y. Cui, and S. Yun. Indoor follow me drone. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '17. ACM, 2017.
[9] L. Mottola, M. Moretta, K. Whitehouse, and C. Ghezzi. Team-level programming of drone sensor networks. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, SenSys '14. ACM, 2014.
[10] C. E. Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*. Springer, 2003.
[11] J. Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.
[12] C. Suduwella, A. Amarasinghe, L. Niroshan, C. Elvitigala, K. De Zoysa, and C. Keppetiyagama. Identifying mosquito breeding sites via drone images. In *Proceedings of the 3rd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*, DroNet '17. ACM, 2017.
[13] O. G. Sutton. A theory of eddy diffusion in the atmosphere. *Proceedings of the royal society of London. Series A, Containing papers of a mathematical and physical character*, 1932.
[14] A. Van't Hof and J. Nieh. Androne: Virtual drone computing in the cloud. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19. ACM, 2019.
[15] D. Vasisht, Z. Kapetanovic, J.-h. Won, X. Jin, R. Chandra, A. Kapoor, S. N. Sinha, M. Sudarshan, and S. Stratman. Farmbeats: An iot platform for data-driven agriculture. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, 2017.