On-the-fly Mobility Event Detection over Aircraft Trajectories

Kostas Patroumpas
IMSI, Athena Research Center, Hellas
kpatro@imis.athena-innovation.gr

Nikos Pelekis
University of Piraeus, Hellas
npelekis@unipi.gr

Yannis Theodoridis
University of Piraeus, Hellas
ytheod@unipi.gr

ABSTRACT

We present an application framework that consumes streaming positions from a large fleet of flying aircrafts monitored in real time over a wide geographical area. Tailored for aviation surveillance, this online processing scheme only retains locations conveying salient *mobility events* along each flight, and *annotates* them as stop, change of speed, heading or altitude, etc. Such evolving *trajectory synopses* must keep in pace with the incoming raw streams so as to get incrementally annotated with minimal loss in accuracy. We also develop one-pass heuristics to eliminate inherent *noise* and provide reliable trajectory representations. Our prototype implementation on top of Apache Flink and Kafka has been tested against various real and synthetic datasets offering concrete evidence of its timeliness, scalability, and compression efficiency, with tolerable concessions to the quality of resulting trajectory approximations.

CCS CONCEPTS

• **Information systems** → **Data streaming**;

KEYWORDS

air traffic management, geostreaming, mobility events, trajectories

ACM Reference Format:
Kostas Patroumpas, Nikos Pelekis, and Yannis Theodoridis. 2018. On-the-fly Mobility Event Detection over Aircraft Trajectories. In *26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '18), November 6–9, 2018, Seattle, WA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3274895.3274970

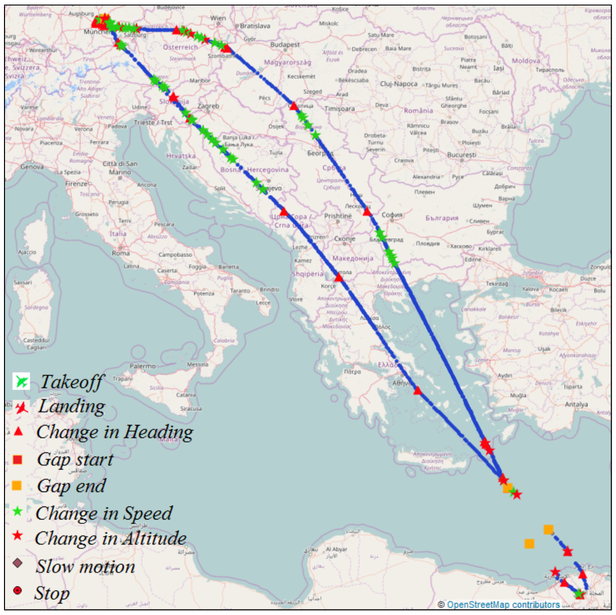


Figure 1: Mobility events detected along a roundtrip flight

1 INTRODUCTION

Each year, commercial airlines allow hundreds of millions of passengers to travel for tourism or business. In Europe only, 26,000 flights take place every day. Air transport is a strategic industry and is expected to grow by 5% annually until 2030[★], raising concerns about safety and its potentially serious impact on the already congested skies. High competition and slim profit margins due to labor and fuel costs, force airlines to carefully plan their schedules, routes, maintenance services, crew rotations and ticket prices.

Air Traffic Management (ATM) systems employ *flight plans* as the best representation of the intended route of aircrafts so as to minimize direct operating costs. ATM systems necessitate accurate and predictable *aircraft trajectories* and traditionally employ a *model-based* approach, but this has certain limitations related to inherent inaccuracies, diverse sources of error and the effect of uncertainty (weather, traffic, etc.). Yet, modern technology enables real-time tracking of aircrafts, either via *Automatic Dependent Surveillance – Broadcast* (ADS–B) messages emitted by aircrafts or through monitoring by terrestrial radar stations. So, a *data-driven* approach to trajectory management may be advantageous to more efficient ATM. With numerous aircrafts in the air, their flight plans may often need adjustment due to traffic congestion or emergency situations, so timely and reliable trajectory representation for all ongoing flights in a wide geographical area is crucial.

Towards this goal, we propose a novel application framework for *real-time detection of mobility events* over aircraft trajectories. We cope with voluminous, fluctuating, and noisy surveillance streams from large fleets of aircrafts reporting positions at varying frequencies. We apply single-pass techniques to obtain succinct, lightweight representation of trajectories without harming approximation quality. Except for adverse weather conditions, traffic regulations, congestion or manoeuvres (a.k.a. holding patterns) near airports, aircrafts expectedly follow almost straight, predictable routes in the air. Instead of retaining every incoming location, we propose to drop any predictable positions along trajectory segments of "normal" motion characteristics. As illustrated in Figure 1, we only keep *critical points* conveying salient mobility events (annotated as stop, slow motion, change in heading, change of speed, change of altitude, etc.) identified when the motion pattern of a given aircraft changes

[★]https://ec.europa.eu/transport/modes/air_en significantly. This process also involves eliminating inherent *noise* in the streaming positions due to delayed messages, duplicate positions, crosswind drift, discrepancies in GPS measurements, etc.

Thus, a large amount of raw positional updates can be suppressed with minimal loss in accuracy. This derived stream of *trajectory synopses* must keep in pace with input streaming positions so as to get incrementally annotated with important mobility events. Such compressed trajectories can be particularly useful in efficient online or offline post-processing (e.g., mobility analytics, statistics, pattern mining, etc.), and facilitate their comparison irrespectively of differing frequencies of updates among aircrafts.

Overall, we address the following major challenges in managing positional data streams in aviation surveillance:

– *Timeliness.* Mobility event detection must be carried out in real-time. Critical points concerning evolving trajectories of aircrafts must be issued at *operational latency* (ideally within milliseconds, or at most a few seconds) to enable immediate action if necessary.
– *Compression.* Trajectory synopses retain only notable mobility events per aircraft and thus incur a small memory footprint. At least 70% of the raw data in our tests was discarded as redundant; and in case of frequent updates, compression attains 99%.
– *Quality.* Such light-weight representations are reliable enough in reconstructing trajectories with small deviations (i.e., tolerable approximation error) from original traces, also coping with inherent stream imperfections (network delays, noise, etc.).
– *Scalability.* Not only do we verify that a centralized deployment of our framework can handle scalable volumes of incoming positions at varying arrival rates, but we also demonstrate that it can be also parallelized, thus offering even higher gains in efficiency.

To the best of our knowledge, this is the first trajectory-aware application framework specifically tailored for real-time surveillance over noisy, intermittent, geostreaming messages in aviation. Our contribution can be summarized as follows:

- We prescribe a processing flow to detect important mobility events online, and thus incrementally maintain succinct, reliable representations of aircraft trajectories (Section 3).
- We introduce single-pass spatiotemporal rules and noise reduction heuristics against streaming positions from numerous flying aircrafts, with significant benefits in compression and quality of the resulting trajectory synopses (Section 4).
- We implement a prototype on Apache Flink [2] and Kafka [3] that can sustain massive position updates from a fleet of aircrafts and promptly detect changes in their mobility (Section 5).
- We empirically validate our prototype on performance and approximation quality with simulations against real datasets, as well as synthetically enlarged data in scalability tests (Section 6).

2 RELATED WORK

Stream processing platforms like Apache Flink [2], Spark [4], or Storm [5] offer powerful customizable and extensible capabilities to ingest, process, and aggregate massive streams from diverse Big Data sources. An overview and experimental study of their current processing capabilities are discussed in [13] with a particular focus on analytics. Besides, extensions to support spatial representation and indexing, as well as topological queries have emerged for several Big Data platforms, like SpatialHadoop [24], GeoSpark [11], or Simba [27]. However, they focus on point locations and simple spatial features, ignoring entirely any spatiotemporal notions concerning movement and lacking support for trajectory management. Recently, UlTraMan [10] introduces support for storing and querying historical trajectories over Spark. In contrast, we aim to handle evolving trajectories *online*, hence we opt for Apache Flink as stream processing engine and we add custom functionality.

Our approach on trajectory synopses over positional streams essentially involves a kind of *online* path simplification. Hence, batch (offline) algorithms like [17, 19] are ruled out, since they require knowledge of all locations in advance. In online methods, retained samples should keep each compressed trajectory as much closer to the original one, chiefly by minimizing approximation error as in trajectory fitting techniques [7, 18]. For instance, a sliding window approach in [18] keeps discarding points until the error exceeds a given threshold. The STTrace algorithm in [23] uses the concept of safe areas to keep samples that deviate from predefined speed and direction error bounds. Dead-reckoning policies like [26] and mobility tracking protocols in [14] can be employed on board of vehicles to relay positional updates only upon significant deviations in their known course. A bounded quadrant system suggested in [16] enables estimation of various error bounds for trajectory compression in ageing-aware fashion suitable for tracking devices of limited storage. The one-pass, error-bounded algorithm in [15] involves local distance checking and optimizations to achieve higher compression. Although such generic data reduction techniques [28] could be applied on streaming aircraft trajectories, they entirely lack support for mobility-annotated features in the retained samples.

In [21, 22] we introduced a maritime surveillance platform specifically for tracking vessel trajectories and also recognizing complex events (e.g., suspicious vessel activity). Applying a sliding window over the streaming positions, that technique reported any detected events periodically (i.e., upon window slides) with all recent "delta" changes at a compression ratio better than 95% over the raw data. Our current approach differs substantially, and not just because it deals with aircrafts, i.e., locations with an extra *z*-ordinate; this is not trivial, as it involves handling additional events (e.g., change in altitude, takeoff, landing) and extra filters for noise elimination. Mobility events with richer semantics (multiple annotations per location) can now be emitted at minimal latency, instead of relying on the slide step of windows. Finally, our new prototype is specifically designed for scalable execution in modern cluster infrastructures.

Our approach is also distinct from either deterministic or probabilistic techniques (e.g., [6]) towards *predicting* aircraft trajectories (i.e., flight plans). Instead, we aim at a data-driven, online summarization and semantic annotation of *evolving* trajectories, retaining only salient information in a streaming fashion. To the best of our knowledge, no other approach associates extra *annotations* like stop, slow motion, turn, etc. to the incoming locations. Apart from data compression, our approach can be further advantageous in post-processing. Though segmentation, these annotated features may yield *semantic trajectories* [20] for more advanced interpretation of mobility patterns. Via suitably defined rules, our primitive mobility events could also act as notifications to timely trigger detection of more *complex events* [9], such as holding patterns near airports, aircrafts in danger of collision, sudden deviations from flight plans, etc., analogous to those applied by CEP-traj [25] over vessel traces.

3 FRAMEWORK SPECIFICATIONS

Figure 2 depicts the processing flow of our framework. Input from surveillance streams arrives as messages carrying 4-d timestamped positions $p = \langle x, y, z, \tau \rangle$ of aircrafts. Messages may be admitted via external connectors to online surveillance sources or replayed from historical files using custom data feeders from publish-subscribe systems (e.g., Apache Kafka [3]). Without loss of generality, we assume that altitude values z are expressed in feet, whereas (x, y) represent GPS longitude/latitude coordinates of known georeference (e.g., WGS1984). Timestamp values τ are considered as discrete, totally-ordered time instants, e.g., UNIX epochs in milliseconds.

The **Noise Reduction** module provides distinct sequences of timestamped positions per aircraft, after excluding any inherent *noise* detected in the streaming positions due to e.g., delayed or duplicate messages, GPS discrepancies, etc. Thus:

Definition 1. For a given aircraft o, its *trajectory* T_o is a possibly unbounded, time-ordered sequence of tuples $\langle o, p_i \rangle$. Each successive timestamped position p_i consists of (x_i, y_i, z_i) coordinates in 3-d Euclidean space recorded at timestamp τ_i.

The resulting *noise-free locations* are issued as a derived stream (marked #1 in Figure 2). The m latest of such positions *per aircraft* are maintained as its persistent *state* by the **Mobility States** module. This enables calculation of its mean velocity vector and other spatiotemporal estimates (rate of turn, rate of climb, etc.).

The **Mobility Event Detection** module tracks changes along each aircraft's path, taking advantage of its mobility state to avoid false alarms. For each aircraft, it digests a subset of its noise-free locations as characteristic positions to incrementally maintain its approximate *trajectory synopsis*. By effectively discarding redundant locations along a "normal" course, this process instantly identifies "critical points" to denote *mobility events* (stop, slow motion, change in altitude, etc.) along each trajectory. Currently, each aircraft is examined *in isolation* from the rest; interaction among aircrafts towards richer mobility semantics is left for future work. A detected event may yield one, two or multiple critical points (e.g., the start and end of a slow motion event), but always in a streaming fashion (indicated as derived stream #2 in Figure 2). Formally:

Definition 2. *Synopsis* S_o over trajectory T_o of aircraft o consists of a possibly unbounded, time-ordered sequence of *critical points*, each represented as a tuple $\langle o, p_i, A_i \rangle$, where $\langle o, p_i \rangle \in T_o$ and A_i is a set of *annotations* denoting mobility events at position p_i.

For a given aircraft o, it holds that $S_o \subset T_o$; ideally, such summarization should provide a concise synopsis with $|S_o| \ll |T_o|$, but also minimal deviation from the original trajectory. Set A_i denotes *multiple annotations* for a single timestamped location p_i, acting like a bitmap where setting a bit signifies that a particular mobility event (e.g., a change in heading) was detected at that position.

Besides, this module also signals out possible disruptions in communication. Typically in real-world surveillance streams, an aircraft may cease relaying positions for quite a while (e.g., flights over oceans). Importantly for mission-critical or emergency situations, its last reported location marks such a *communication gap* and is emitted into a derived stream of *notifications* (denoted as #3 in Figure 2). Currently, we make no attempt to "fill-in" such gaps either via interpolation, extrapolation, or historical patterns.

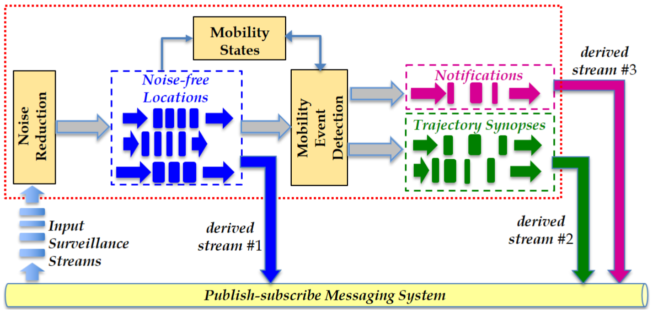


Figure 2: Online processing flow of the framework

Both noise reduction and mobility event detection on trajectories are controlled by a user-specified *parametrization*. Choosing proper parameter values strongly depends on dataset characteristics and should be fine-tuned in order to trade compression efficiency with quality of results, as detailed in our empirical validation in Section 6.

In the sequel, we suppose that a fresh tuple $\langle o, p_{cur} \rangle$ reports p_{cur} as the latest known location of a given aircraft o, whereas p_{prev} is its previously relayed position at time $p_{prev}.\tau < p_{cur}.\tau$. In detecting significant motion changes per aircraft, we employ an *instantaneous velocity vector* $\overrightarrow{v}_{cur}$ over its two latest positions p_{prev}, p_{cur}. In addition, we maintain the *mean velocity* $\overrightarrow{v_m}$ per aircraft to get its short-term course based on m recent positions in its mobility state. In trajectory computations, we typically employ linear interpolation [7] using Haversine distances, given that any two consecutive positions (p_{prev}, p_{cur}) are generally close enough.

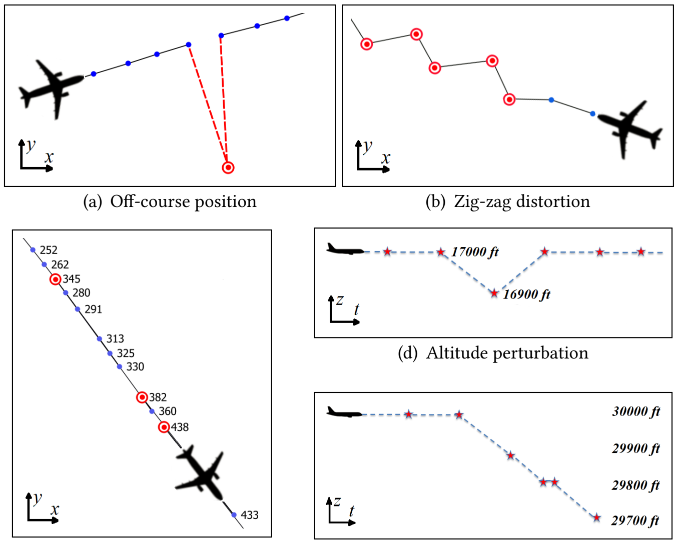
4 ONLINE PROCESSING OVER STREAMING AIRCRAFT TRAJECTORIES

Next, we present the main modules in our framework, concerning noise reduction from aviation surveillance streams (Section 4.1), maintenance of mobility states (Section 4.2), and the rules guiding detection of mobility events along aircraft trajectories (Section 4.3).

4.1 Online Noise Reduction

Despite their high value in aircraft surveillance, positional data streams are not error-free, particularly *ADS-B messages* relayed from aircrafts. Apart from *spurious coordinates* indicating impossible positions across a flight, also *aircraft identifiers* may be invalid, inexistent or duplicate. Besides, *timestamps* are assigned upon reception at a base station (or radar), so are subject to unsynchronized clocks. Messages collected from a network of stations may not be properly deduplicated (e.g., a message received by multiple stations) and their timestamps may not be adjusted; thus, ordering of positions is distorted. In addition, satellite transmission problems may lead to *delayed* or *missing* messages. There may be also *glitches in altitude* values, usually measured by multiples of 100 feet. As aircrafts fly at high speeds, *granularity of timestamp values* should be the finest possible (in milliseconds). But in real-world aviation data, several seconds may pass before receiving another fresh position from an aircraft, exacerbating deviations in spatiotemporal estimations, especially altitude-related ones (e.g., rate of climb).

Avoiding costly offline cleansing, we apply *online, single-pass* empirical filters upon admission of each streaming position. Since we aim at lightweight trajectory representations, we can afford to



(a) Off-course position (b) Zig-zag distortion

(c) Out-of-sequence positions (d) Altitude perturbation (e) Altitude discrepancy

Figure 3: Noise-related situations along aircraft trajectories

lose garbled, out-of-sequence positions and not consider correcting their timestamps, since more fresh locations will arrive soon. Subject to a careful parametrization (Table 1), a noisy situation may be identified if at least one of the following conditions apply:

- *Off-course positions* incur abrupt changes in velocity vector $\overrightarrow{v}_{cur}$. Figure 3(a) illustrates such an outlier with an abnormal deviation from known course as abstracted by mean velocity $\overrightarrow{v_m}$ of the aircraft over its previous m positions. In that case, it is most probable that $v_{cur} > v_{max}$, e.g., exceeding a speed limit of $v_{max} = 1000$ knots is not expected by a commercial aircraft.
- Flying aircrafts normally take a turn smoothly, marked by a series of transmitted locations. So, a position should be dropped if it indicates an abrupt turn (e.g., $\Delta\phi \geq 120^o$) with respect to the known course. In adverse situations (e.g., a storm) a trajectory may look like a *'zig-zag'* with a series of considerable turns as in Figure 3(b). Discarding those consecutive points as noise is not typically correct; yet, in terms of data reduction this is quite desirable, as the aircraft tries to keep to its planned course.
- Sometimes, an aircraft appears to accelerate far too much (over δ_{max} knots/hour), which is rather unusual in aviation. This is typical for *out-of-sequence* messages with twisted timestamps. As shown in Figure 3(c), all locations may be along the flight course, but due to the late arrival of the three red spots to the base station, the aircraft looks like suddenly retracting backwards at an unrealistic speed.
- Identical locations from the same aircraft in the air should be considered as noise. Agitations in GPS readings when an aircraft is stationed are normal, but coincidental coordinates in succession are almost certainly duplicate messages received by multiple base stations and assigned with slightly differing timestamps.
- Distinct messages from a given aircraft may have the same timestamp but report different coordinates. To resolve such conflicts in timestamping, we arbitrarily keep one of these messages.
- Upon a sudden surge in the *rate of turn*, e.g., exceeding a given threshold $\rho_{max} \geq 10^o$/sec, the current location is dropped as noise. Very sharp turns are unusual for large-sized airplanes.
- To filter out *discrepancies in altitude*, we take advantage of the mobility state maintained per aircraft over its latest m positions. Computing a moving average in the rate of climb generally eliminates false indications regarding altitude change, as the one in Figure 3(d). These are mainly caused by the lack of precision in altitude measurements usually reported at discrete flight levels (multiples of 100 feet). This rule can also filter out small deviations in altitude not actually qualifying as mobility events as shown in Figure 3(e), where two consecutive locations are reported at 29800 feet while the aircraft has clearly started its descent.
- Another case of noise concerns ascents or descents at a rate more than γ_{max} (e.g., 200 feet/sec) between successive locations. We discard such temporary steep changes in altitude, usually caused by false timestamping or imprecise altitude measurements.
- An aircraft cannot be stopped in the air, so it cannot have a speed $v_{cur} \cong 0$ at an altitude above a threshold h_0 (e.g., 100 feet); under h_0 an aircraft may be safely considered as landed to the ground.

Table 1: Parameters for noise reduction

Symbol	*Threshold description*	*Default value*
v_{max}	Max speed of movement for any aircraft	1000 knots
δ_{max}	Max rate of change in speed	50000 knots/h
$\Delta\phi$	Max difference between successive headings	120^o
ρ_{max}	Max rate of turn between successive positions	10^o/sec
γ_{max}	Max rate of climb between successive positions	200 feet/sec
h_0	Max altitude to consider an aircraft as landed	100 feet

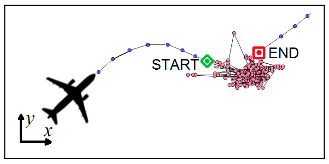
As we experimentally verified (cf. Section 6), noise may concern up to 13% of received positions, qualifying to any of the aforementioned cases, so data cleaning is a necessary step before any further processing of aircraft trajectories. Crucially, accepting noisy positions would drastically distort the resulting synopses (as in Figure 3(a)), and also hamper proper detection of mobility events.

4.2 Mobility State Maintenance

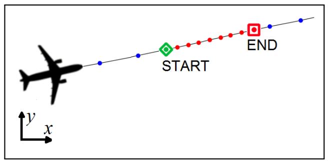
To assist mobility event detection, the most recent portion of each evolving trajectory T_o is continuously maintained as a sequence of m noise-free, chronologically ordered position updates $\langle o, p_i \rangle$. We call this trajectory portion the current *mobility state* for aircraft o, maintained in memory as a *count-based* sliding window ranging over its m recent locations; m is a small integer (e.g., $m = 10$), chosen according to the reporting frequency of aircrafts. From information available in this state, mean velocity vector $\overrightarrow{v}_m$ of that aircraft can be calculated, as well as several spatiotemporal estimates (such as distance, travel time, overall change in heading, rate of turn, rate of climb, etc.). To avoid considering obsolete locations in computations, we also set a maximum timespan of ω units (e.g., 15 minutes in the past) for positions retained in the state.

Table 2: Mobility events detected along aircraft trajectories

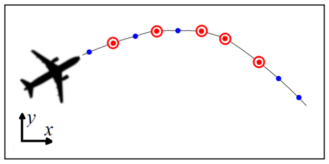
	Mobility Event	*Parameter*	*Critical point(s)*
(x, y)	Stop	v_{min} (knots)	Start point & End point
	Slow Motion	v_θ (knots)	Start point & End point
	Change in Heading	$\Delta\theta$ (^{o})	One or multiple points
	Change of Speed	α (%)	Start point & End point
	Communication Gap	ΔT (sec)	Start point (*notifications*) & End point (*synopses*)
z	Change in Altitude	$\Delta\gamma$ (ft/sec)	One or multiple points
	Takeoff	h_0	Start point
	Landing	h_0	Start point



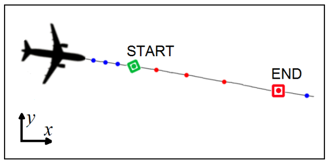
(a) Stop



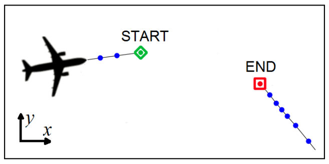
(b) Slow motion



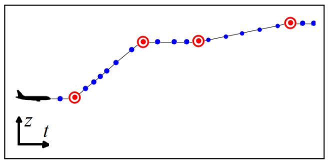
(c) Changes in heading



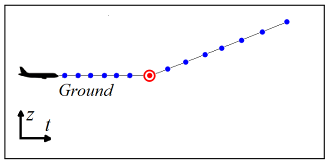
(d) Change of speed



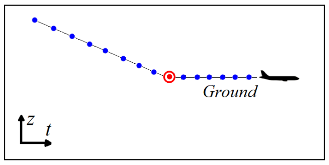
(e) Communication gap



(f) Changes in altitude



(g) Takeoff



(h) Landing

Figure 4: Mobility events (of one, two, or multiple critical points) detected along the trajectory of a flying aircraft

4.3 Mobility Event Detection

Next, we specify intuitive heuristics to capture mobility events along aircraft trajectories. As depicted in Table 2, some types of events occur on the (x, y) plane (e.g., stops or changes in heading), while others in the z dimension, like changes in altitude. Depending on the type of a mobility event, one, two or multiple critical points may be issued to fully describe it. For example, stops are characterized by their start and end points to indicate the duration of this event. Instead, when a significant change in heading is observed, then a single critical point marks this location as a turning point in the trajectory; and in case of smooth turns, a series of such critical points may be emitted in order to better capture this event.

Stop indicates a stationary aircraft (e.g., at airport gates) over a period of time, so actually several nearby positions may be received due to GPS discrepancies. To capture its duration, two critical points should be emitted as shown in Figure 4(a):

Start of stop annotates a fresh location once $v_{cur} \leq v_{min}$,

where v_{min} is an appropriately chosen speed threshold for denoting immobility (e.g., 1 knot). Any subsequent locations also validating this condition are not emitted as critical with respect to stop, implying that this phenomenon is still ongoing. Hence:

End of stop is identified once $v_{cur} > v_{min}$ *and* also this aircraft was previously stopped (i.e., a *Start of stop* had been issued).

Slow Motion indicates an aircraft moving at a very low speed below a threshold v_σ (e.g., 5 knots) over some time. Essentially, this concerns a subtrajectory on the ground, e.g., while taxiing on runways. Provided that sufficient position reports exist (scarcely the case in real-world data), the first and the last positions in this subsequence (Figure 4(b)) should be emitted as critical points:

Start of slow motion in case that $(v_{cur} \leq v_\sigma) \wedge (v_{prev} > v_\sigma)$.

Subsequent positions also having a speed below v_σ are not emitted as critical with respect to slow motion, since this pattern still holds.

End of slow motion is issued *either* when the aircraft stops (i.e., concurrently with a *Start of stop* annotation) *or* it holds that $(v_{cur} > v_\sigma) \wedge (v_{prev} \leq v_\sigma)$, so it no longer moves slowly.

Change in Heading may be spotted according to a given *threshold angle* $\Delta\theta$; e.g., if there is a difference of $\Delta\theta > 10^o$ between the current and previous bearing values. To avoid false alarms caused by occasional GPS errors, we detect such changes in heading by checking for large deviations between instantaneous velocity $\overrightarrow{v}_{cur}$ from mean velocity $\overrightarrow{v}_m$ as retained in the aircraft's state:

Change in Heading is detected once $arccos(\frac{\overrightarrow{v}_{cur} \cdot \overrightarrow{v}_m}{\|\overrightarrow{v}_{cur}\| \|\overrightarrow{v}_m\|}) \geq \Delta\theta$.

Although such a change is detected upon receiving current location p_{cur}, the *previous* one p_{prev} must be actually annotated as critical point; this offers a more precise trajectory approximation, as at p_{cur} the aircraft already moves along a changed course.

In addition, we also need to check whether the *cumulative* change of instantaneous headings across a few previous positions exceeds threshold $\Delta\theta$ (as we applied in [21] for vessels). As illustrated in Figure 4(c), a series of such critical points offers a more reliable piecewise approximation of the actual flight, as aircrafts usually make smooth turns or manoeuvres. Of course, the lesser the angle threshold $\Delta\theta$, the more the critical points denoting such slight changes in heading. The last critical point in the series denotes that the aircraft is about to follow a straight course afterwards.

Change of Speed occurs once current speed v_{cur} deviates by more than α% from mean speed v_m. Given a parameter α, which quantifies a cutoff threshold for ignoring occasional perturbations in speed estimations due to GPS error, wind drift, etc., a pair of critical points (Figure 4(d)) should be annotated:

Start of speed change occurs once it holds $|\frac{v_{cur}-v_m}{v_m}| > \alpha$.

Subsequent positions where this condition still holds (i.e., the aircraft keeps speeding up or slowing down) are not emitted as critical with respect to speed change, as long as the aircraft is considered to continue moving by the same pattern.

End of speed change occurs once $|\frac{v_{cur}-v_m}{v_m}| \leq \alpha$ *and* a critical point has been earlier annotated as *Start of speed change*.

This end point indicates that aircraft speed practically stabilizes close to the mean v_m based on its mobility state. Such events may be further distinguished into *acceleration* or *deceleration*, effectively by taking the sign (+/−) of the fraction in the aforementioned rules.

Communication Gap occurs often in long-haul flights over oceans when contact with base stations is lost. As illustrated in Figure 4(e), when no message is received from an aircraft for at least a *time interval* ΔT, e.g., over the past 10 minutes, its actual course is unknown. Such gaps might be also caused by accidents or other suspicious situations (e.g., hijacks). Obviously, this event is concluded once contact is restored, sometimes after a period much longer than ΔT. To maintain *trajectory synopses* in append-only fashion at operational latency, current location p_{cur} is promptly annotated as

End of communication gap if $(p_{cur}.\tau - p_{prev}.\tau > \Delta T)$

by checking with the time elapsed from previous location p_{prev}.

Still, a *notification* may be also issued concerning the location at which contact was lost, i.e., position p_{prev} is annotated as

Start of communication gap once $(\tau_{now} - p_{prev}.\tau > \Delta T)$.

However, such notifications *cannot* be issued promptly, but with at least ΔT time units delay from system time τ_{now}. To avoid disrupting temporal ordering of other critical points in trajectory synopses, notifications are derived as a *separate stream* (Figure 2).

Change in Altitude identifies a transition to a different cruise level, as depicted in Figure 4(f). Due to occasional spurious jumps in reported altitudes (Figure 3(d)), we avoid checking directly against altitude values. Instead, we detect positions with significant change at the *rate of climb* (or *descent*) γ of the aircraft. This is the vertical speed (in feet/sec) between successive locations when ascending (respectively, descending):

$$\gamma = \frac{p_{cur}.z - p_{prev}.z}{p_{cur}.\tau - p_{prev}.\tau}.$$

Our goal is to identify positions where changes in this rate start or cease to take place. On first sight, by setting a *threshold* $\Delta\gamma > 0$ for significant changes in this rate (e.g., more than 20 feet/sec), a critical point should be issued once it is found that $|\gamma_{cur}| > \Delta\gamma$, where γ_{cur} expresses the rate of climb (or descent) of the aircraft at timestamp $p_{cur}.\tau$. As with changes in heading, note that actually the *previous* location p_{prev} must be annotated as critical, because at current location p_{cur} the aircraft is already on its ascent (or descent). In addition, we should avoid characterizing as critical all locations along such a climb (or descent), as long as there is no significant change in the previously calculated rate γ_{prev}. Moreover, to avoid emitting potentially invalid critical points caused by numeric imprecision in timestamp and altitude values, we may conservatively detect such changes by also checking with the *mean rate of climb* (or *descent*) γ_m computed over the m most recently reported locations of the aircraft available in its mobility state. Overall, we propose to annotate location p_{prev} as

Change in Altitude if $(|\gamma_{prev}| \leq \Delta\gamma < |\gamma_{cur}|) \wedge (|\gamma_{cur} - \gamma_m| \geq \Delta\gamma)$.

Note that a series of such critical points may need be issued to approximate a smooth ascent (or descent) more reliably.

Takeoff concerns the last location along a flight that reports the aircraft still on the ground. To overcome discrepancies in measurements, altitude values should be compared against a threshold h_0 denoting the highest altitude at which an aircraft may be considered as landed (e.g., $h_0 = 100$ feet). Thus,

Takeoff is issued once $(p_{cur}.z > h_0) \wedge (p_{prev}.z \leq h_0)$.

Note that previous location p_{prev} gets annotated as takeoff, so as to provide a better trajectory approximation (Figure 4(g)).

Landing concerns the location at which an aircraft touches the ground (Figure 4(h)). With a rationale similar to takeoffs, we check against threshold h_0, so current location p_{cur} is annotated as

Landing in case that $(p_{cur}.z \leq h_0) \wedge (p_{prev}.z > h_0)$.

5 PROTOTYPE IMPLEMENTATION ON FLINK

We implemented an application prototype using the DataStream API of *Apache Flink* [2] with *Apache Kafka* [3] as broker for streaming messages. Flink adopts a *tuple-at-a-time* model for its pipelined stream operators, offering results at low latency and achieving higher throughput [8, 12] compared to the *micro-batch* approach in Spark Streaming [4]. Flink also supports *exactly-once* semantics for fault tolerance, guaranteeing that each tuple will be processed to ensure correct results. This is particularly important for our application, as losing tuples may completely alter spatiotemporal estimates (e.g., distance, speed) and distort identification of events.

5.1 Custom Support for Trajectory Management

Flink currently lacks native support for *spatial entities* (e.g., points, polygons), let alone *spatiotemporal* moving objects and related methods (speed, heading, acceleration, etc.). Thus, we had to introduce custom data structures in Scala for maintaining trajectories and support all mobility operations required in our processing flow (Section 4). We also specified an extensible *attribute schema* in Avro [1] with all spatiotemporal properties involved in aircraft trajectories. This is used to hold all attributes in the original messages (identifiers, locations, timestamps, indications of various aircraft sensors, etc.), but also spatiotemporal properties dynamically calculated during our processing (e.g., travelled distance, rate of climb, etc.). Further, we defined specific *topics* in Kafka, for handling messages from input surveillance sources and the derived streams (Figure 2).

Flink offers powerful *partitioning* features for its operators. It automatically distributes streaming items by a key (in our case, this is a unique identifier per aircraft or flight) into parallel instances *of each operator*, each maintaining its own local state. In our case, this feature caused more trouble than benefit, exactly because it is established at operator level. Indeed, our processing strategy needs to maintain a mobility state *per aircraft*, i.e., its most recent trajectory locations. Once a fresh position is admitted, it must be processed by a pipeline of operators (Figure 5) that actually encapsulate our custom methods. Thus, we need to keep a *separate persistent state per aircraft* across all operators in the entire pipeline, and not rely on operator states. As Flink did not provide any flexibility to maintain a globally accessible state, we resorted to implementing a custom hashing scheme in Scala. Based on aircraft identifiers (*key*), it keeps a small number m of recent locations per aircraft (its *mobility state*), as well as certain flags that immediately reflect its current status (e.g., is stopped, in slow motion, has changed speed). Of course, this incurs some overhead (once a fresh location arrives, the state must be updated) and housekeeping (discard obsolete items), but otherwise trajectory-aware processing would not have been possible.

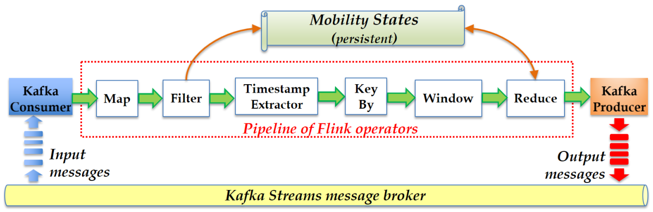


Figure 5: Pipeline of utilized Flink and Kafka operators

Most importantly for real-world applications, Flink provides support for *event time* semantics, i.e., using timestamps generated by the data sources. In our case, timestamp values in aircraft positions are assigned upon reception to a base station or when captured by radars. In real-world aviation surveillance streams, *delayed messages* are not the exception, but rather the norm. In order to tolerate delayed messages from aircrafts in the pipeline, we specify a *lag* value for maximum delays (say, 30 seconds) behind the latest known timestamp and accordingly reorder messages chronologically.

5.2 Processing through a Pipeline of Operators

Figure 5 illustrates the pipeline of Kafka and Flink operators utilized in our prototype implementation, each working as follows:

Kafka Consumers: To simulate streaming input into our application, we have built data feeders in Scala that can parse historical records of aircraft traces and push positional messages to a specified Kafka topic. Records are replayed either according to their *original* arrival rate or consumed by a user-specified *fixed* stream rate (e.g., 10000 messages/sec). A custom consumer subscribes to a particular Kafka topic and starts receiving timestamped stream messages.

Map: Each incoming message is mapped to the attribute schema. Afterwards, it travels through the pipeline as a tuple; annotations may be assigned upon checking by following operators.

Filter: Discards noisy locations, which are not further considered. In case that a fresh location concerns an aircraft currently not being monitored, its *mobility state* is initialized; otherwise, this location is appended to the existing state evicting the oldest location therein.

Timestamp Extractor: We employ *time watermarks* in Flink to allow processing of delayed messages up to a given *lag* value.

Keyed Windowing: Flink offers several types of *windows*, which split a stream into temporary finite "pieces" used in computations. We employed *count-based sliding windows* over the latest two tuples *keyed by* aircraft identifier, in order to calculate instantaneous spatiotemporal estimates (distance, heading, rate of turn, altitude change, etc.) between consecutive locations. This way, spatiotemporal computation is streamlined with the arrival of each fresh message and the resulting estimates can be assigned to the tuple as it is being processed by all subsequent operators in the pipeline.

Reduce: The core of our application logic is encapsulated in a series of custom functions under this generic operator. Essentially, Reduce takes a pair of consecutive locations (p_{prev}, p_{cur}) of a given aircraft and triggers evaluation of the detection rules in Section 4. If a location qualifies as a critical point for an event, the respective bit in its annotation is set; eventually, multiple bits may be set.

Kafka Producers: Output is promptly pushed into distinct Kafka topics by three producers, each dealing with a derived stream (*noise-free raw locations, trajectory synopses, notifications*).

Table 3: Datasets used in the simulations

Dataset (O: original; S: synthetic)	*#messages*	*#air-crafts*	*Reports every*	*Time-span*	*Spatial coverage*
(*O*) FlightAware ADS-B	3701966	24416	68 sec	1 day	Europe
(*O*) ADSBHub ADS-B	4291620	5516	28 sec	1 day	Europe
(*O*) EnAire IFS tracklogs	24180100	36939	5 sec	1 week	Spain
(*S*) FlightAware ADS-B	50019756	13591	1 sec	6 hours	Europe
(*S*) EnAire IFS tracklogs	16165671	5066	1 sec	1 day	Spain

6 EMPIRICAL VALIDATION

In this Section, we report experimental results from an extensive validation of our prototype against aviation surveillance streams.

6.1 Experimental Setup

Our prototype runs entirely in main memory to efficiently cope with volatile, asynchronously updated, streaming locations from thousands of aircrafts. Apart from its *centralized* execution with one instance handling all incoming data, we also examined its *parallelization* for advanced efficiency against scalable arrival rates. By design, each trajectory is examined in isolation from the rest. Hence, in parallelization tests we employ a varying number of concurrent threads, each monitoring a disjoint subset of aircrafts.

In these tests, it is not the total volume of input data that matters, but the *arrival rate* of position updates, i.e., how frequently each aircraft reports a fresh position. As this is actually rather low in all available *original* aviation data, we also created *synthetic* datasets at a very high rate in order to verify robustness and scalability of the prototype. Datasets simulated in our study (Table 3) include:

– *FlightAware original ADS-B dataset.* Aircraft positions over Europe were collected via the FlightAware API[1] during one full day and were replayed in our simulations at their original arrival rate; on average, each aircraft reports its position every 68 seconds.
– *ADSBHub original ADS-B dataset.* Aircraft positions over Europe were collected via ADSBHub[2] during one full day. Again, we performed simulations at the original arrival rate; on average, each aircraft relays a position message every 28 seconds.
– *EnAire original IFS radar tracklogs.* Aircraft positions over Spain were tracked by International Flight Support secondary radars during a week (courtesy: Air Navigation Service Provider EnAire[3]). Locations of flying aircrafts are captured every 5 seconds on average, admittedly the best stream rate among original datasets.
– *FlightAware synthetic ADS-B dataset.* For a subset of FlightAware data during a 6-hour interval, we inflated each flight with extra positions at *every second* via interpolation between each pair of consecutive raw positions. We avoided taking samples when aircrafts are not flying or are out of contact, by setting the maximum timespan between successive original locations to 5 minutes.
– *EnAire synthetic IFS radar tracklogs.* We applied a similar interpolation over positions in EnAire original IFS logs on a full day.

Evidently, the online detection process is sensitive to *parametrization*, which is a trade-off between reduction efficiency and approximation accuracy. Table 4 lists such parameters used in our tests (default values in bold). For their suitable calibration, apart from consulting domain experts, we conducted exploratory tests on subsets of available surveillance data. If data reduction is the primary

[1]https://flightaware.com/commercial/data/adsb/
[2]http://www.adsbhub.org/
[3]https://www.enaire.es/services

Table 4: Parameters involved in mobility event detection

Symbol	*Description*	*Values*
υ_{min}	Minimum speed for asserting movement	**1** knot
υ_{σ}	Maximum speed for asserting slow motion	**5** knots
$\Delta\theta$	Threshold for asserting change in heading	2.5^o, $\mathbf{5^o}$, 7.5^o,10^o
α	Minimum rate for asserting speed change	**10%**
ΔT	Minimum time interval for asserting gaps	1, **2**, 5, 10 minutes
$\Delta\gamma$	Threshold for asserting changes on rate of climb	10, **20**, 30 feet/sec
m	Positions retained in each aircraft's state	**5**, 10

goal, a more relaxed parametrization should be applied (e.g., a larger angle $\Delta\theta$ to keep fewer turning points) without much harm in accuracy. With more strict thresholds, additional mobility events can be detected, thus capturing slighter changes along trajectories but also reducing the compression ratio in the resulting synopses.

Next, we report indicative results from an extensive experimental study with parametrizations as listed in Table 1 and Table 4. Simulations were conducted on a machine running Linux Ubuntu 16.04.2 LTS with Intel® Xeon® 24-core CPU at 3.07GHz and 48GB RAM. Unless otherwise specified, performance results on time and throughput are averaged over the entire duration of each stream. Compression ratio and approximation quality are averages over all trajectories throughout their varying timespans in each dataset.

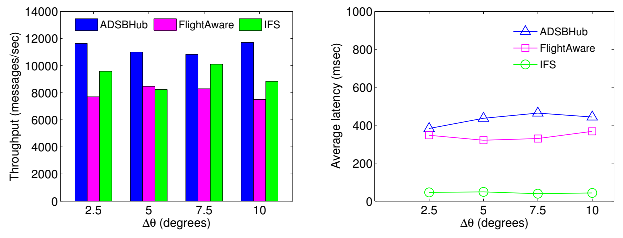
6.2 Experimental Results

6.2.1 Timeliness. In these simulations against original data at their original arrival rate, we measure *throughput*, i.e., the amount of input messages per second that may be handled by the prototype concerning the entire operator pipeline. We also measure *average latency*, i.e., the average time that a tuple remains in the pipeline since its admission until it is issued in a derived stream (or discarded as noise). These performance results concern a *centralized* instance (i.e., single thread) of the prototype without any data partitioning.

Figure 6 illustrates results with a varying angle threshold $\Delta\theta$ (affecting detection of turning points); results by varying other parameters are similar. It is clear from Figure 6(a) that even on a single-threaded instance, our framework can effectively cope with several thousand messages/sec. Fluctuations in throughput among datasets should be attributed to the differing number of aircrafts, as well as their diverse reporting frequencies. Besides, critical points are emitted at operational latency (in milliseconds) in all cases (Figure 6(b)). Thanks to the increased arrival rate in IFS radar tracklogs, mobility events are detected almost instantly (less than 40ms on average). The more frequently an aircraft reports an update, the more rapidly a potential critical point may be issued. Even with the other two datasets of inferior sampling rates, latency always remains at operational level (<500ms), although clearly increased. By and large, these tests confirm that our prototype can timely detect mobility events for thousands of aircrafts in real-world scenarios even without parallelization.

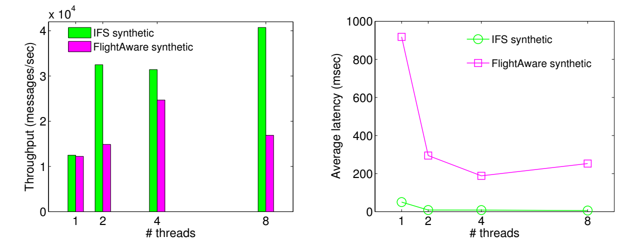
6.2.2 Scalability with Increased Arrival Rates. For a more stringent assessment of the prototype in terms of scalability, we performed extra simulations against synthetic datasets (Section 6.1).

The fact that the framework maintains each trajectory in isolation from the rest, offers great opportunities for parallelization. So, we employed a varying number of *concurrent threads*, each one monitoring a distinct subset of aircrafts according to a simple data partitioning scheme. For simplicity, this subdivision is based on



(a) *throughput* (b) *average latency*

Figure 6: Timeliness against *original* datasets (single thread)



(a) *throughput* (b) *average latency*

Figure 7: Scalability over *synthetic* data (multiple threads)

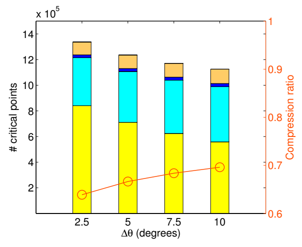
simple hashing over the identifier of each aircraft, such that its positions are always propagated to the same thread so as to consistently establish connected trajectory segments.

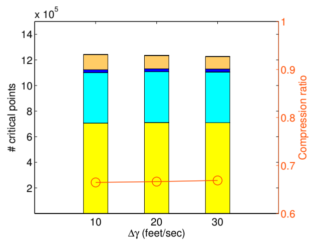
Quite expectedly, latency is far better when multiple threads are employed, as plotted in Figure 7(b). With respect to throughput (Figure 7(a)), higher performance of up to 40,000 messages/sec may be achieved via data partitioning. Regarding the interpolated FlightAware dataset, note that throughput drops with 8 threads. This is mostly caused by imbalanced assignment of aircrafts into partitions, as our hash-based scheme ignores other important factors, such as duration of flights, arrival rate per partition, etc. But, even this simplified data partitioning confirms that the prototype is capable of handling scalable volumes of streaming positions and has great potential for parallelization and advanced load balancing.

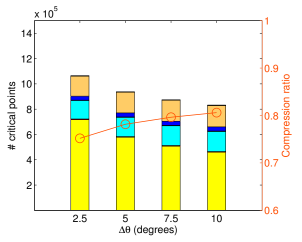
6.2.3 Compression Efficiency. As every data reduction process, effectiveness of trajectory summarization can be assessed via its *compression ratio*. This is the percentage (%) of positions dropped from the approximate trajectory synopses over the raw ones originally obtained, or equivalently:

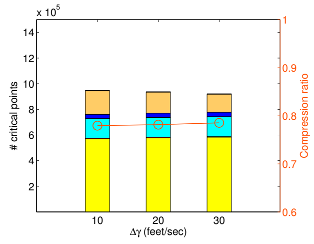
$$compression\ ratio = 1 - \frac{\#critical\ points}{\#original\ positions}.$$

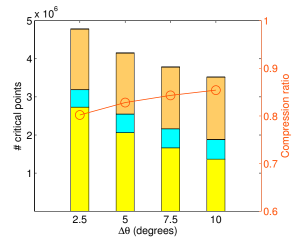
The higher this ratio, the more lightweight the resulting synopses. A ratio close to 1 implies stronger data reduction, as the vast majority of raw locations are dropped and few critical points suffice to represent each trajectory. The red line in the plots of Figure 8 depicts this ratio with varying parameters so as to quantify their effect on compression. In particular, we examine how compression varies with angle threshold $\Delta\theta$ (affecting changes in heading) and threshold $\Delta\gamma$ on rate of climb (capturing changes in altitude). Bar plots show breakdowns of annotated critical points and how much each event type weighs in trajectory synopses, since variations in parameters affect detection for certain types of mobility events.


(a) FlightAware: *varying angle*


(b) FlightAware: *varying rate of climb*


(c) ADSBHub: *varying angle*


(d) ADSBHub: *varying rate of climb*


(e) IFS: *varying angle*

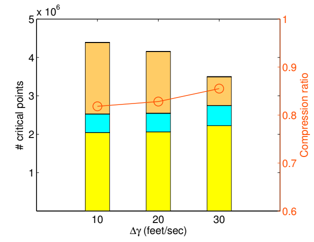

(f) IFS: *varying rate of climb*

Figure 8: Compression ratio over *original* datasets

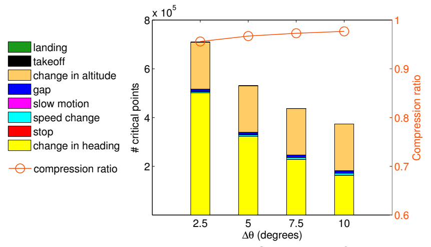


Figure 9: Compression ratio for *synthetic IFS* data

Figures 8(a), 8(c), and 8(e) illustrate the effect of varying angle thresholds on trajectory summarization against the three original datasets. Quite expectedly, relaxing parameter $\Delta\theta$ leads to increased compression, since less turning points get detected. Actually, the achieved compression ratio reflects the reporting frequency in the original data sources. Messages from ADSBHub and FlightAware arrive less frequently, so they have more chances to indicate an important change in mobility (particularly in heading, speed, or altitude) and thus be singled out as critical. In IFS logs, positions are reported more often, so many more may get discarded as "normal" leading to an increased compression close to 85%.

A varying threshold $\Delta\gamma$ on the rate of climb seems to have negligible impact on compression for ADS-B data, as shown in Figures 8(b) and 8(d). Indeed, sharp fluctuations on this rate are scarce given the infrequent updates from those aircrafts. In contrast, with IFS tracklogs (Figure 8(f)) the amount of critical points concerning altitude keeps diminishing considerably with a more relaxed threshold. The reason is twofold and has to do with precision in the original data. First, altitudes are measured in hundreds of feet, so even slight perturbations (as in Figure 3(d)) may incur changed indications. Moreover, timestamps in IFS logs have a granularity of seconds and not milliseconds; given the relatively high frequency of updates (every 5 sec), this may yield sudden spikes in the rate of climb.

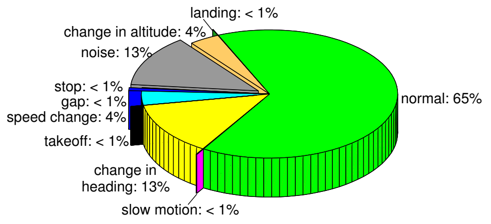
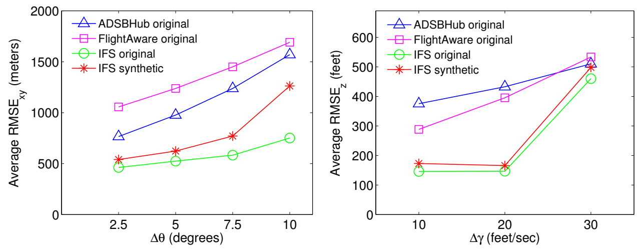


Figure 10: Annotated positions in *original* ADSBHub dataset

Despite relaxed parametrizations, compression ratio over *original* datasets is rather moderate (never exceeds 85%) because of the generally low arrival rate in position updates. Thus, we also measured compression ratio over *synthetic* (interpolated) data consistently yielding a reduction over 95% (Figure 9). Intuitively, when positions are relayed more often, a larger proportion of them expectedly qualifies as "normal", and hence can be safely excluded from the synopses. The detected critical points are practically those identified also in the original dataset, as they concern the same events along trajectories (changes in altitude, heading, etc.). Typically, small variations in compression ratio depend on parametrization, as indicatively shown with angle tolerance $\Delta\theta$. But overall, we can clearly conclude that with a higher reporting frequency in the original surveillance streams, compression would have been more dramatic, soaring close to 99% as this experiment testifies.

Figure 10 depicts a breakdown of annotations assigned to all locations in the original ADSBHub dataset (similar results were obtained with the other two datasets). About two thirds of those positions are considered "normal" and thus suppressed as redundant from trajectory synopses. There is also a lot of noise (13%) that must be discarded altogether from raw surveillance data (we also observed a lesser, yet significant noise in the other two data sources). But what is really striking is the almost complete lack of mobility events on the ground (i.e., stop, slow motion, takeoff, landing), something definitely caused by insufficient raw positions in airport areas. In total, we identified 847 takeoffs and 2129 landings in this dataset, but this is far from ideal, given that tens of thousands of flights take place across Europe daily. It turns out that, unless more streaming positions become available in airport areas, such events cannot be safely captured with a fair possibility.

6.2.4 Approximation Quality. Preserving only critical points in trajectory synopses incurs a lossy approximation. To assess the quality of such compressed trajectories, we estimated their deviation from original ones, by computing the pairwise Haversine distance H on (x, y) plane between *synchronized* locations from an original trajectory (composed of noise-free locations) and its synopsis (consisting of critical points). If an original location p_i at timestamp τ_i is not critical, then its corresponding time-aligned p'_i in the synopsis is estimated via linear interpolation along the path that connects the



(a) *$RMSE_{xy}$ on location* (b) *$RMSE_z$ on altitude*

Figure 11: Quality of trajectory approximation

two critical points before and after τ_i. Similarly, for each original altitude z_i reported at timestamp τ_i from an aircraft, we estimate via linear interpolation a time-aligned altitude z'_i in its trajectory synopsis. Finally, for each aircraft that reported M raw positions in total, we estimated the *Root Mean Square Error* between the original and synchronized sequences of its locations on the (x, y) plane and over its respective altitudes in the z dimension:

$$RMSE_{xy} = \sqrt{\frac{1}{M} \cdot \sum_{i=1}^{M} (H(p_i, p'_i))^2}\ , \quad RMSE_z = \sqrt{\frac{1}{M} \cdot \sum_{i=1}^{M} (z_i - z'_i)^2}.$$

For each trajectory, $RMSE_{xy}$ is in meters, while $RMSE_z$ is in feet.

We take averages of these error measures over each trajectory dataset. A rather surprising remark from Figure 11(a) is that RMSE_{xy} is considerable. By cross-checking flights and their respective synopses on the (x, y) plane, we found that their linear paths are almost identical. But due to inherent flaws in timestamp precision and reporting frequency, discrepancies in speed estimates are inevitable, so a time-aligned location p'_i lies either ahead or behind w.r.t. original p_i along the same path. Error is much less in original IFS data (updated every 5 seconds), but is still considerable even for the synthetic IFS data, so more frequently relayed positions cannot remedy the problem alone. More precision is required in timestamp values (as well as on locations and altitude) to cope with fast moving aircrafts and promptly catch significant mobility events. Anyway, this error is less than a nautical mile (≅1852 meters) and can safely ensure horizontal separation of flights, significantly lower than the limit of 3 miles specified by international regulations.

Quality of synopses in the z dimension is far better, as testified in Figure 11(b). Thanks to a moderate reporting frequency in IFS data, it seems that our rules can successfully capture most, if not all, changes in altitude events. But error deteriorates with ADS-B data, mostly because of the lack of precision (by 100 feet) in altitude values. With a finer precision, we could expect significant decrease in this error, minimizing information loss and improving the fidelity of synopses. Yet, such error on altitude estimates is more than acceptable, as international aviation standards specify a vertical separation between commercial flights at 1000 feet.

7 CONCLUSIONS AND FUTURE WORK

We introduced a novel application framework that can identify mobility events online along evolving trajectories from large fleets of aircrafts. So, trajectory synopses can retain only important changes in their movement with significant savings in space and processing costs. Performance results indicate that our prototype achieves low latency and tolerable error, while efficiently coping with scalable volumes of positional updates from aviation surveillance streams.

Deployment in a distributed cluster infrastructure is under way, expecting significant performance gains. We intend to explore adaptive detection and auto-calibrated parametrization of events, also correlating with contextual information (weather, air traffic control areas, etc.). Finally, handling other mobility data streams (e.g., vessels, merchandise) may be possible with suitable rule specifications.

ACKNOWLEDGMENTS

This work has been partially supported by the EU-funded *datAcron* project (H2020-ICT-2015-687591). The datasets used in simulations are courtesy of Boeing Research & Technology Europe and EnAire.

REFERENCES

[1] Apache Avro data serialization. https://avro.apache.org/docs/current/.
[2] Apache Flink. https://flink.apache.org/.
[3] Apache Kafka distributed stream platform. https://kafka.apache.org/.
[4] Apache Spark Streaming. http://spark.apache.org/streaming/.
[5] Apache Storm. http://storm.apache.org/.
[6] S. Ayhan and H. Samet. Aircraft trajectory prediction made easy with predictive analytics. In *KDD*, pages 21–30, 2016.
[7] H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *VLDB Journal*, 15(3):211–228, 2006.
[8] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache Flink: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38:28–38, 2015.
[9] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, 2012.
[10] X. Ding, L. Chen, Y. Gao, C. S. Jensen, and H. Bao. UlTraMan: A unified platform for big trajectory data management and analytics. *PVLDB*, 11(7):787–799, 2018.
[11] GeoSpark. http://geospark.datasyslab.org/.
[12] S. Hagedorn, P. Götze, and K. Sattler. Big spatial data processing frameworks: Feature and performance evaluation. In *EDBT*, pages 490–493, 2017.
[13] A. Kipf, V. Pandey, J. Böttcher, L. Braun, T. Neumann, and A. Kemper. Analytics on fast data: Main-memory database systems versus modern streaming systems. In *EDBT*, pages 49–60, 2017.
[14] R. Lange, F. Dürr, and K. Rothermel. Efficient real-time trajectory tracking. *VLDB Journal*, 20(5):671–694, 2011.
[15] X. Lin, S. Ma, H. Zhang, T. Wo, and J. Huai. One-pass error bounded trajectory simplification. *PVLDB*, 10(7):841–852, 2017.
[16] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, J. Lee, and R. Jurdak. A novel framework for online amnesic trajectory compression in resource-constrained environments. *IEEE Trans. Knowl. Data Eng.*, 28(11):2827–2841, 2016.
[17] C. Long, R. C.-W. Wong, and H. Jagadish. Trajectory simplification: On minimizing the direction-based error. *PVLDB*, 8(1):49–60, 2014.
[18] N. Meratnia and R. de By. Spatiotemporal compression techniques for moving point objects. In *EDBT*, pages 765–782, 2004.
[19] J. Muckell, P. Olsen, J.-H. Hwang, C. Lawson, and S. Ravi. Compression of trajectory data: A comprehensive evaluation and new approach. *Geoinformatica*, 18(3):435–460, 2014.
[20] C. Parent, S. Spaccapietra, C. Renso, G. Andrienko, N. Andrienko, V. Bogorny, M. L. Damiani, A. Gkoulalas-Divanis, J. Macedo, N. Pelekis, Y. Theodoridis, and Z. Yan. Semantic trajectories modeling and analysis. *ACM Comput. Surv.*, 45(4):42:1–42:32, 2013.
[21] K. Patroumpas, E. Alevizos, A. Artikis, M. Vodas, N. Pelekis, and Y. Theodoridis. Online event recognition from moving vessel trajectories. *GeoInformatica*, 21(2):389–427, 2017.
[22] K. Patroumpas, A. Artikis, N. Katzouris, M. Vodas, Y. Theodoridis, and N. Pelekis. Event recognition for maritime surveillance. In *EDBT*, pages 629–640, 2015.
[23] M. Potamias, K. Patroumpas, and T. Sellis. Sampling trajectory streams with spatiotemporal criteria. In *SSDBM*, pages 275–284, 2006.
[24] SpatialHadoop. http://spatialhadoop.cs.umn.edu/.
[25] F. Terroso-Saenz, M. Valdés-Vela, E. den Breejen, P. Hanckmann, R. Dekker, and A. F. Skarmeta-Gómez. CEP-traj: An event-based solution to process trajectory data. *Information Systems*, 52:34–54, 2015.
[26] O. Wolfson, A. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distrib. Parallel Dat.*, 7(3):257–287, 1999.
[27] D. Xie, F. Li, B. Yao, G. Li, L. Zhou, and M. Guo. Simba: Efficient in-memory spatial analytics. In *SIGMOD*, pages 1071–1085, 2016.
[28] D. Zhang, M. Ding, D. Yang, Y. Liu, J. Fan, and H. T. Shen. Trajectory simplification: An experimental study and quality analysis. *PVLDB*, 11(9):934–946, 2018.