

OBSERVABILITY-FIRST ENGINEERING FOR REAL-TIME ADVERTISING PIPELINES: LATENCY REGRESSION DETECTION AND REVENUE PROTECTION

Sujan Kumar Reddy Challa ¹

Received 21.07.2026.
Revised 06.09.2026.
Accepted 02.10.2026.

Keywords:

Observability, Service Level Objectives, Real-Time Bidding, Distributed Systems Monitoring, Anomaly Detection, Digital Advertising Systems.

Original research

ABSTRACT

Real-time advertising pipelines, particularly those supporting connected television (CTV) and video real-time bidding (RTB), operate under strict latency budgets while processing high volumes of heterogeneous events across bid requests, bid responses, fills, and downstream delivery signals. Generic application performance monitoring tools, built for request-response web services, frequently fail to capture the specific failure modes of these pipelines: latency regressions that erode fill rate before triggering conventional error alerts, and partner integrations that go silent rather than fail loudly. This paper presents a practitioner-oriented observability framework structured around three complementary pillars: a metric design approach tailored to the granularity and cardinality demands of RTB telemetry; a service-level-objective-based alerting architecture using multiple time windows and burn rates to reduce false positives; and a dashboard pattern that correlates technical signals with revenue impact for both engineering and business audiences. The paper also proposes treating the absence of expected data, such as a demand-side platform that stops responding without generating errors, as a distinct failure class requiring dedicated instrumentation. The framework draws on established observability, site reliability engineering, and microservice fault-analysis research, adapted to the specific constraints of advertising infrastructure, and is illustrated with hypothetical worked examples and validated through a discrete-event simulation study on synthetic RTB telemetry (Section 9). It is intended as a practical reference for engineering teams operating latency-sensitive, revenue-linked distributed systems.



© 2026 Journal of Engineering, Management and Information Technolog

1. INTRODUCTION

A real-time advertising pipeline links publishers, ad exchanges, and demand-side platforms through a sequence of exchanges bound tightly by time: a bid request goes out, DSPs must answer within a strict window, a winning bid gets picked, and the ad either delivers or the impression goes unfilled. CTV inventory

in particular tends to operate under tight latency budgets set by the exchange, so any regression in that budget can suppress bid participation, cut fill rate, and take revenue with it (Ou et al., 2024).

Standard application performance monitoring assumes trouble announces itself, usually through elevated error rates or thrown exceptions. Advertising pipelines break that assumption twice over. A latency regression can hurt

¹ Corresponding author: Sujan Kumar Reddy Challa
Email: sujan.challa10@gmail.com

revenue long before it crosses any error threshold, since a slow-but-technically-successful response may simply land too late to count. And a DSP partner can go quiet altogether, producing no error signal at all on the exchange side, a silent failure that dashboards built around error rate and request count routinely miss.

This paper proposes a three-pillar framework for teams running RTB and CTV advertising infrastructure: metric design fit for the granularity advertising telemetry actually needs, an SLO-based alerting architecture using multi-window, multi-burn-rate techniques, and a dashboard pattern here called the Revenue Protection Dashboard that ties technical indicators directly to business impact. A fourth idea threads through the paper: the absence of expected data deserves treatment as its own failure class, separate from conventional error-based detection. What follows reviews related work, sketches the pipeline context, walks through each pillar with illustrative worked examples, and closes with adoption considerations and limitations.

1.1 Scope and Approach

The contribution offered here is a conceptual framework and set of design patterns, not a report of new empirical measurements. It was developed by synthesizing findings from three adjacent research literatures, observability and distributed tracing, site reliability engineering and alerting theory, and microservice fault and failure-detector analysis, and adapting their concepts specifically to the constraints of real-time advertising pipelines, a domain that existing literature in those three areas does not address directly. The framework's mechanics are demonstrated through hypothetical worked examples rather than production traffic; in response to peer review, the metric-segmentation approach of Section 4, the alerting approach of Section 5, the paired-panel pattern of Section 6, and the silence-detection approach of Section 7 are additionally validated through a simulation study on synthetic traffic, reported in Section 9. Every quantitative figure that is not explicitly marked hypothetical or reported as a simulation result is drawn verbatim from a cited external source. This scoping is stated explicitly here so that the sections that follow can be read as a design-pattern contribution on its own terms, rather than judged against the evidentiary standard of an empirical systems paper.

2. RELATED WORK

Observability engineering has grown up considerably as microservice architectures have taken over from monolithic deployments. Surveys of observability in containerized and edge-distributed systems trace the move from centralized logging toward the combined use of logs, metrics, and traces as the three pillars of system visibility (Usman et al., 2022). More recent work pushes that taxonomy further, sorting observability frameworks by deployment paradigm, comparing agent-based and

agentless instrumentation strategies, and pointing to persistent difficulty correlating signals across service boundaries once a system spans more than a handful of independently deployed services (Faseeha et al., 2025). Neither survey addresses domains where a monitoring signal is directly and immediately convertible into a dollar figure, which is the gap this paper's Revenue Protection Dashboard pattern is intended to fill.

Distributed tracing has been used directly for anomaly detection in microservice cloud applications, flagging deviations from trace-level features in real time (Raeiszadeh et al., 2023); related work applies tracing and service-mesh telemetry to dependency analysis inside Kubernetes deployments, showing that non-intrusive, sidecar-based instrumentation can reconstruct service dependency graphs without requiring application-level code changes (Wang et al., 2022). A separate strand of research picks up once an anomaly has already been flagged, surveying the shift from statistical correlation methods toward causal and graph-based root-cause localization and cataloging the tradeoffs between interpretability and detection accuracy across these approaches (Fu et al., 2025), and cataloging detection and root-cause techniques specific to service-based cloud applications, including both threshold-based and machine-learning-based anomaly scoring (Soldani & Brogi, 2022). This paper does not attempt root-cause localization once an anomaly is detected; it instead concentrates on the earlier problem of designing metrics and alerts so that a revenue-relevant anomaly is surfaced in the first place, an emphasis that complements rather than duplicates this line of work (Duarte et al., 2023).

Failure detection theory offers a useful angle on the absence-of-data problem this paper raises later. Foundational work established the unreliable failure detector abstraction for asynchronous distributed systems, formally characterizing the accuracy and completeness properties a monitoring process must satisfy to distinguish a crashed component from one that is merely slow (Chandra & Toueg, 1996). Building on that abstraction, more recent work on distributed system-level diagnosis has tried to reconcile classical diagnosis models with unreliable failure detectors, formally working out the tradeoffs between detection latency and accuracy for scalable, decentralized monitoring topologies (Duarte et al., 2023). Separately, industrial fault analysis of microservice systems has found that a large share of production incidents trace back to configuration and environment rather than application logic, a finding drawn from a multi-year industrial survey and benchmark study (Zhou et al., 2021), with later work proposing parallelized delta-debugging to localize such faults faster by exploring candidate root causes concurrently rather than sequentially (Zhou et al., 2022). Neither the theoretical failure-detector line of work nor the industrial fault-analysis studies cited above were developed with a revenue-bearing, request-level timeout in mind, which is the specific condition this paper's Section 7 treats as a first-class design constraint rather than a secondary property of the monitored system.

Log-based anomaly detection has been studied as a complement to metrics and traces. Empirical comparisons of deep learning and classical approaches suggest the two do not perform consistently differently once evaluation choices such as data grouping and class balance are controlled, even though several deep learning models report an F-measure exceeding 0.9 on the widely used HDFS benchmark dataset under favorable evaluation conditions (Le & Zhang, 2022; Yu et al., 2024). Nearby work on alerting itself matters here too: research on alert fatigue in security operations centers documents how excessive or poorly prioritized alerting wears down operator response quality over time, drawing on both survey data from working analysts and simulated alert-stream experiments (Tariq et al., 2025), and machine-learning-based alert suppression has demonstrated accuracy exceeding 90 percent in reducing operator-facing alert volume in cloud monitoring deployments, offering one path to easing this burden without simply raising alert thresholds across the board (Voutsas et al., 2024).

Within advertising specifically, RTB research has mostly focused on bid optimization and auction strategy rather than pipeline reliability. Surveys of bid optimization summarize algorithmic progress in response to shifting auction dynamics, covering both classical constrained-optimization formulations and more recent reinforcement-learning-based bidding agents (Ou et al., 2024). This paper's contribution sits next to that body of work rather than inside it: instead of optimizing bidding strategy, it addresses the observability and reliability engineering needed to keep the pipeline itself inside its latency and availability budget, a question existing RTB research touches only indirectly, typically as a fixed constraint assumed by the optimization model rather than as an object of study in its own right.

3. SYSTEM CONTEXT: REAL-TIME ADVERTISING PIPELINE ARCHITECTURE

A typical CTV RTB pipeline starts when a publisher's ad server or supply-side platform issues a bid request for an available impression. That request goes out to multiple DSPs at once, each of which has to evaluate the opportunity and answer inside a fixed timeout, often tens to a couple hundred milliseconds depending on the exchange. The exchange picks a winning bid, and the creative delivers to the viewer.

Several architectural details make this sequence more demanding than it first appears. First, most modern exchanges run parallel, header-bidding-style auctions rather than a single sequential waterfall, meaning a given impression opportunity may be broadcast to dozens of DSPs simultaneously, and the pipeline's aggregate outbound and inbound request volume scales with that fan-out rather than with impression volume alone. Second, CTV inventory frequently arrives in pods,

meaning a single ad break may require several impressions to be auctioned and filled in quick succession, so a latency regression affecting one auction within a pod can cascade into missed fill for subsequent auctions in the same break if the pipeline does not isolate per-auction timing budgets. Third, exchanges and DSPs are commonly distributed across multiple geographic regions to keep round-trip time within budget for viewers in different markets, which means the same nominal timeout can represent very different amounts of headroom depending on which regional path a given request traverses.

Table 1. Pipeline stage flow for a CTV RTB transaction, represented in tabular form (author's own analysis)

Stage	Actor	Action
Bid request	Publisher ad server / SSP	Issues bid request for an available impression
Bid evaluation	DSPs (parallel)	Evaluate the opportunity and answer within a fixed timeout (tens to ~200 ms)
Auction	Exchange	Runs parallel, header-bidding-style auctions and picks the winning bid
Delivery	Exchange / publisher	Serves the winning creative to the viewer

Every stage in that sequence is a place where latency can build up, and because the whole chain runs against a hard timeout, a delay anywhere can knock out a bid that would otherwise have won (Table 1). A slow web request degrades user experience, but a late RTB bid response is typically discarded outright, converting a latency problem into a lost-revenue event almost immediately. That tight coupling between latency and revenue is the central justification for treating observability as core engineering work in this domain, rather than an afterthought bolted on once something breaks.

4. PILLAR ONE: METRIC DESIGN FOR RTB PIPELINES

Good metric design for RTB pipelines means picking indicators fine-grained enough to expose partner-level and geography-level variation, without generating so many unique time series that the monitoring backend buckles under the cardinality. Five metrics form a workable core: bid request rate, bid response rate, fill rate, latency percentiles, and revenue per thousand impressions (RPM) (Table 2).

Table 2. Core RTB metric taxonomy

Metric	Definition	Granularity	Revenue Linkage
Bid request rate	Volume of bid requests issued per unit time	Per DSP, per inventory segment	Leading indicator of available opportunity
Bid response rate	Share of requests receiving a valid response	Per DSP, per region	Precedes fill-rate change
Fill rate	Share of opportunities resulting in a filled impression	Per device type, per geography	Direct technical proxy for revenue
Latency percentiles (p50/p95/p99)	Distribution of end-to-end response time	Per DSP, per pipeline stage	Tail latency drives timeout-related losses
RPM	Revenue per thousand impressions	Per inventory source	Direct business outcome metric

Bid request and response rates, tracked per DSP and per inventory segment, tend to expose participation trends before fill rate itself moves. Fill rate, the share of opportunities that actually get filled, is the most direct technical stand-in for revenue and should be broken out by device type and geography, since CTV inventory often behaves quite differently from mobile or desktop under otherwise identical pipeline conditions.

Latency is better tracked as percentiles, typically p50, p95, and p99, rather than as an average, since averages can hide exactly the tail behavior that causes bid timeouts in the first place. High-cardinality dimensions such as DSP identifier, region, and device type are necessary for diagnosis but drive up the number of unique time series considerably; research on time series management systems lays out the tradeoff between keeping full dimensional detail and the storage and query cost that comes with it, which is a useful frame for deciding which dimensions actually deserve full-resolution tracking (Jensen et al., 2017). In practice, this often means retaining full per-dimension detail for a short retention window, sufficient to support real-time alerting and recent-incident diagnosis, while aggregating older data to coarser dimensional detail for longer-term trend analysis, a compromise that keeps recent troubleshooting capability intact without requiring indefinite storage of the highest-cardinality view.

5. PILLAR TWO: SLO-BASED, MULTI-WINDOW MULTI-BURN-RATE ALERTING

Threshold alerting, where a single fixed value trips a page, tends to perform poorly against advertising telemetry because of how naturally volatile it is across time of day, region, and campaign mix. A sturdier approach defines a service-level objective, say a target share of bid responses returned within budget over a rolling window, and alerts on how fast the associated error budget is burning rather than on any one metric crossing a static line.

Multi-window, multi-burn-rate alerting checks burn rate across more than one window at once, usually a short window to catch sudden spikes and a longer one to confirm the consumption is sustained rather than a blip (Table 3). The intent is to cut down false positives triggered by brief, self-correcting fluctuations while still catching genuine regressions quickly, a balance that a single static threshold simply cannot strike, since it has no way to tell a momentary dip from a sustained trend.

Table 3. Burn-rate alerting window comparison, illustrative configuration

Window Pair	Illustrative Burn Rate	Use Case
1 hour / 6 hour	Short window high, long window moderate	Confirms a spike is sustained before paging
6 hour / 3 day	Both windows moderately elevated	Detects slow-building regressions missed by short windows
5 minute / 1 hour	Short window very high, long window low	Flags sudden spikes without paging on self-correcting blips

Applying this to RTB telemetry means being careful about how the underlying SLO is framed. A latency SLO defined against average response time fits poorly here, since it's the tail of the distribution that actually determines timeout behavior; SLOs should instead be pinned to a latency percentile, with burn-rate math run against budget consumption relative to that percentile rather than against raw latency numbers.

For illustration, consider a hypothetical CTV bid-response path with an SLO target of 99.5 percent of responses returned within a 150 millisecond budget over a rolling 30-day period, giving an error budget of 0.5 percent. Suppose the short window, set at 1 hour, shows a burn rate of roughly 14 times the sustainable rate, while the long window, set at 6 hours, shows a burn rate of roughly 6 times. A multi-window rule requiring both windows to exceed their respective thresholds before paging would fire in this hypothetical case, since sustained consumption is confirmed across both windows, whereas a brief 5-minute spike reaching a similar short-window burn rate without a matching long-

window trend would not page, avoiding an unnecessary interruption for a fluctuation that corrects itself. This example is illustrative only, intended to demonstrate the mechanics of the technique rather than to represent a measured production outcome.

Choosing the specific window pairs and burn-rate thresholds is itself a design decision that depends on how volatile a given pipeline's baseline traffic is. A pipeline with a small number of high-volume DSP partners can typically tolerate tighter windows, since aggregate metrics for each partner are less prone to sampling noise; a pipeline spanning many low-volume partners may need wider windows or partner-level aggregation before burn-rate calculations become statistically meaningful, since a single partner's low request volume can otherwise produce false spikes that a larger aggregate would smooth out.

6. PILLAR THREE: REVENUE PROTECTION DASHBOARD PATTERN

Technical dashboards built for engineers and business dashboards built for revenue reporting usually live in separate tools, which slows down the moment a technical regression gets recognized as a revenue event. The Revenue Protection Dashboard pattern proposed here is a single view pairing each core technical metric with its matching revenue indicator, so a latency regression and its downstream effect on fill rate and RPM sit on the same panel instead of requiring someone to cross-reference two separate systems (Table 4).

Table 4. Revenue Protection Dashboard panel pairings, represented in tabular form

Technical panel	Revenue panel	Shared segment
Latency percentile (p50/p95/p99)	Fill rate	DSP segment
Fill rate	RPM trend	Inventory source

The pattern is built around paired panels: a latency percentile panel next to a fill-rate panel for the same segment, and a fill-rate panel next to an RPM trend for the same inventory source. The idea is that an engineer spotting a technical anomaly can immediately see its business consequence, and a business stakeholder can trace an RPM dip back to where it actually started, without either side needing to learn the other's usual tools.

For illustration, suppose the p95 latency panel for a given DSP segment shows a shift from roughly 80 milliseconds to roughly 130 milliseconds over a 20-minute period. On a conventional technical dashboard, this shift might sit unnoticed among dozens of per-service latency panels. On the Revenue Protection Dashboard, the same panel sits paired with the fill-rate panel for that segment, which

in this hypothetical scenario shows a corresponding drop, and with the RPM panel for the associated inventory source, which shows a matching downward trend. Viewing the three paired panels together lets an engineer and a revenue stakeholder recognize, within the same view, that a latency shift is coinciding with a measurable revenue signal, rather than requiring the two audiences to separately notice and then manually correlate two unrelated dashboards. As with the alerting example above, these figures are hypothetical and intended only to demonstrate how the pattern is meant to function.

Dashboard design work in adjacent fields has shown that poorly organized monitoring views slow down recognition of important events and contribute to alert fatigue among the people responsible for responding (Tariq et al., 2025; Voutsas et al., 2024) The Revenue Protection Dashboard pattern tries to avoid this by deliberately limiting scope to a small number of paired, business-relevant panels rather than trying to cram every available metric onto one screen.

7. ABSENCE-OF-DATA AS A DISTINCT FAILURE CLASS

Error-rate monitoring assumes a failing component will generate something observable: an exception, a non-200 response, an explicit timeout. A DSP partner that quietly stops responding, returning nothing rather than an error, may not trigger any of these signals at all, especially if the exchange's own handling code treats a missing response the same as a routine no-bid.

Table 5. Failure class comparison: error-based versus silence-based.

Dimension	Error-Based Failure	Silence-Based Failure
Observable signal	Exception, non-200 response, explicit timeout	No signal at all: absence of expected volume
Typical detection method	Error rate thresholds, exception tracking	Expected-vs-observed volume comparison, heartbeat monitoring
Revenue impact pattern	Partial degradation of a fraction of requests	Full removal of partner's bid volume from auction
Typical on-call recognition speed	Fast: matches existing runbooks	Slow: often mistaken for reduced demand

Catching this failure mode means comparing observed data volume against an expected baseline rather than waiting for an explicit error signal, sometimes called negative-space or heartbeat-based monitoring (Table 5). This echoes, in a practical sense, the distinction classical

failure-detector theory draws between crash faults a monitor can observe directly and faults that have to be inferred from an expected signal simply not arriving within a bounded window (Chandra & Toueg, 1996; Duarte et al., 2023; . Applied to advertising pipelines, that means instrumenting expected-versus-observed bid response volume per partner, rather than leaning solely on response codes or exception counts.

This failure class deserves separate attention because its revenue impact can be proportionally worse than a comparable error-based failure: a partner that silently drops out takes its entire bid volume with it rather than just degrading a slice of requests, and because nothing errors out, on-call response can lag well behind what's achievable for conventional error-triggered alerts.

8. DISCUSSION AND PRACTICAL IMPLICATIONS

Adopting this framework doesn't require ripping out existing observability tooling; the three pillars are meant as a design pattern layered on top of standard metrics, tracing, and alerting infrastructure, not a new platform to stand up. The bigger shift required is organizational: the Revenue Protection Dashboard pattern in particular only works if engineering and business stakeholders agree on a shared set of paired panels, which in practice takes coordination beyond the engineering team alone.

The absence-of-data failure class carries its own process implications too. On-call rotations and runbooks built around error-triggered pages may need explicit extension to cover silence-based alerts, since responders used to diagnosing exceptions may not immediately read a volume-drop alert as a partner outage rather than an ordinary dip in demand-side traffic.

8.1 Implementation Considerations

Teams adopting this framework incrementally will typically find it easier to start with metric re-segmentation than with alerting or dashboard changes, since re-segmenting existing latency and fill-rate metrics by DSP, region, and device type requires no new infrastructure, only a change to how existing telemetry is labeled and aggregated. Multi-window burn-rate alerting can then be layered in once per-partner SLOs are defined and agreed upon, since the technique depends on having a stable, explicit target to measure budget consumption against; attempting burn-rate alerting before an SLO exists tends to produce arbitrary, hard-to-defend thresholds. The Revenue Protection Dashboard pattern is best introduced last, once the underlying metric segmentation and alerting are trustworthy enough that pairing them meaningfully will not simply propagate noisy signals into a business-facing view.

Instrumentation choices should favor open, vendor-neutral telemetry data models over proprietary formats where practical, since advertising pipelines frequently span multiple internal teams and, in some architectures,

multiple cloud environments, and a portable telemetry format reduces the cost of later consolidating dashboards or migrating monitoring backends. This paper does not endorse or evaluate any specific commercial monitoring product, in keeping with its intent as a vendor-neutral design pattern rather than a product recommendation.

9. SIMULATION-BASED VALIDATION OF THE FRAMEWORK

To address the evidentiary gap noted during peer review, this section reports a discrete-event simulation study that exercises all three pillars of the proposed framework against conventional baselines: the per-partner metric segmentation of Section 4, the multi-window, multi-burn-rate alerting approach of Section 5, the expected-vs-observed volume comparison of Section 7, and the paired-panel revenue-protection pattern of Section 6. All data in this section is synthetic; no production traffic was used. The study is described at a level of detail intended to make the experiments reproducible, and every parameter choice is disclosed below.

9.1 Simulation Setup

The simulator models five DSP partners exchanging bid traffic with an exchange at one-minute granularity over 14 simulated days (20,160 minutes). Per partner per minute, bid-request volume is drawn from a Poisson distribution whose mean follows a diurnal curve (peak-to-trough ratio of approximately 2.9 to 1), and each response latency is drawn from a lognormal distribution calibrated so that baseline per-partner p99 latency sits between roughly 110 and 130 ms. A response is counted "bad" when it exceeds a 150 ms budget. The SLO under test mirrors the Section 5 example: 99.5 percent of bid responses within 150 ms over a rolling 30-day window, i.e., an allowed bad fraction of 0.005; burn rate is the observed bad fraction divided by 0.005. All experiments use fixed random seeds (master seed 20260930).

Two alerting strategies are compared, both evaluated per partner (the Pillar One segmentation of Section 4 is granted to both, so the alerting technique is the variable under test):

(A) Static threshold (conventional baseline): page when a partner's trailing 5-minute mean p99 latency exceeds 150 ms.

(B) Multi-window, multi-burn-rate (this paper's proposal): page when any of the three window pairs from Table 3 breaches. The numeric thresholds used are: 1-hour burn above 14x with 6-hour burn above 6x (from the Section 5 worked example); 5-minute burn above 30x with 1-hour burn above 6x (spike pair); and 6-hour burn above 3x with 3-day burn above 2x (slow-burn pair, the "moderately elevated" case in Table 3). The latter two threshold sets are choices made for this study and are disclosed as such.

For the silence failure class of Section 7, the proposed expected-vs-observed volume comparison (alert when

observed per-partner response volume falls below 50 percent of the trailing 3-day same-minute-of-day median for 3 consecutive minutes) is compared against a conventional error-rate monitor (page when per-minute explicit error fraction exceeds 2 percent).

Four scenarios are injected into one partner (DSP-3) starting at day 5, 00:00: (S0) no incident, a 14-day clean baseline; (S1) a gradual latency regression with p99 drifting from roughly 115 ms to roughly 200 ms over 6 hours and then sustained; (S2) a self-correcting 5-minute p99 spike to roughly 365 ms; (S3) a 4-hour silent dropout in which the partner stops responding entirely without emitting errors. Scenario S4 reuses the S1 drift with a synthetic revenue model (fill rate = 0.62 - 1.6 x timeout share; RPM proportional to fill rate) to exercise the paired-panel pattern of Section 6.

9.2 Per-Partner Metric Segmentation (Pillar One)

Pillar One (Section 4) argues that RTB telemetry must be segmented per partner and per inventory segment, because pipeline-wide aggregates dilute partner-level regressions below detection thresholds. Scenario S1 tests this claim directly: the same gradual drift is evaluated twice, once with per-partner p99 latency (the paper's proposed granularity) and once with a traffic-weighted pipeline-wide p99 (the aggregate view a team without segmentation would see).

Table 6. Metric granularity comparison on the S1 gradual drift (synthetic data).

Metric view	Detection latency	Missed
Per-partner p99 (segmented)	189 min	no
Pipeline-wide p99 (aggregate)	—	yes

Table 6 shows the result. The segmented view detects the drift 189 minutes after injection, when DSP-3's p99 crosses the 150 ms budget. The aggregate view never detects it: with four healthy partners diluting the signal, the pipeline-wide p99 reaches only 134.4 ms by the end of the drift, comfortably below the 150 ms threshold, even as the affected partner's p99 reaches 200.2 ms. The regression — a real, sustained latency doubling on 20 percent of bid traffic — is invisible in the aggregate.

9.3 Multi-Window, Multi-Burn-Rate Alerting (Pillar Two)

Table 7 summarizes the alerting comparison. Over the 14-day incident-free baseline (S0), neither strategy produced a false page. On the gradual regression (S1), the static threshold paged 189 minutes into the drift, while the multi-window rule paged after 1,298 minutes (approximately 21.6 hours) via the 6h/3d slow-burn pair; the 1h/6h fast-burn pair, tuned for sharper deviations, did not breach on this mild drift. On the self-correcting spike

(S2), the static threshold paged (a false positive on an event that resolved itself), while the multi-window rule correctly stayed silent: the 5-minute burn rate spiked but the confirming 1-hour window did not, so no page fired.

Table 7. Simulation results: static-threshold vs. multi-window, multi-burn-rate alerting (synthetic data, master seed 20260930).

Scenario	Static threshold (A)	Multi-window, multi-burn-rate (B)
S0: 14-day baseline, false pages	0	0
S1: gradual 6-hour drift, detection latency	189 min	1,298 min (via 6h/3d slow-burn pair)
S2: self-correcting 5-minute spike	paged (false positive)	silent (correct)

Sensitivity checks across four additional random seeds reproduced this pattern in every run (zero baseline false pages for both strategies; the static threshold paging on every spike run; the multi-window rule silent on every spike run).

9.4 Silence Failure Class

For the silence failure class (S3), the error-rate monitor never paged during the 4-hour dropout: with no errors emitted, there was nothing for it to trigger on. The expected-vs-observed volume comparison detected the dropout 2 minutes after onset, and this result was likewise stable across seeds (Table 8).

Table 8. Simulation results: silence detection during a 4-hour partner dropout (synthetic data).

Monitor	Detection latency	Missed
Error-rate monitor (>2%/min)	—	yes
Expected-vs-observed volume comparison	2 min	no

9.5 Paired-Panel Revenue Protection (Pillar Three)

Pillar Three (Section 6) claims that pairing technical and revenue panels in one view surfaces revenue impact earlier than alerting alone. Scenario S4 quantifies this lead time. The revenue dip — fill rate declining from 61.6 percent to 58.3 percent and RPM from \$8.44 to \$8.00, approximately a 5.2 percent revenue decline — becomes visible on the paired panels about 350 minutes into the drift, while the slow-burn latency alert does not confirm until 1,298 minutes (Table 9). The dashboard pattern thus surfaces the revenue consequence roughly 948 minutes

(about 15.8 hours) before the alerting system confirms the underlying regression.

Table 9. Paired-panel lead time on the S4 revenue scenario (synthetic data)

Signal	Value
Fill rate: baseline → trough	61.6% → 58.3%
RPM: baseline → trough	\$8.44 → \$8.00
Revenue dip visible on paired panel	350 min after injection
Slow-burn alert confirms regression	1,298 min after injection
Dashboard lead time	~948 min (~15.8 h)

For gradual regressions, the revenue panel is the earlier signal and the alert arrives as confirmation rather than as the first indication — which is exactly the behavior the Revenue Protection Dashboard pattern is designed to support.

9.6 Interpretation

The results support the framework’s three central claims with one important qualification. Per-partner metric segmentation detected a gradual regression that pipeline-wide aggregates missed entirely. Multi-window, multi-burn-rate alerting eliminated the false page on the self-correcting spike that the static threshold produced, and the expected-vs-observed volume comparison detected a silent dropout that error-rate monitoring missed entirely. The paired-panel dashboard surfaced the revenue consequence of a gradual regression roughly 16 hours before the alerting system confirmed it. The qualification is detection latency on gradual regressions: the static threshold was faster (189 vs. 1,298 minutes) on a clean, sustained drift, because the illustrative fast-burn thresholds are tuned for sharper deviations and the slow-burn pair needs a long window to confirm. This is consistent with the paper’s framing of window and threshold selection as a design decision dependent on baseline traffic volatility, and it suggests practitioners should expect to tune the pairs rather than adopt the illustrative values verbatim. All results are conditional on the stated synthetic parameters and should be read as evidence about the mechanisms’ behavior, not as performance guarantees for any production deployment.

10.LIMITATIONS

This paper offers a practitioner framework informed by observability, reliability engineering, and microservice fault-analysis research, rather than an empirical evaluation against production traffic at scale within the paper itself. The specific metric thresholds, burn-rate window pairs, and dashboard layouts described here are illustrative patterns, not validated parameters, and teams

adopting the framework should expect to tune them against their own traffic. The worked examples in Sections 5 and 6 are hypothetical and are included only to demonstrate mechanics, not to report measured outcomes.

The framework also doesn’t address privacy-preserving measurement constraints that increasingly shape identifier-level advertising telemetry, which is left open for future work. Generalizability beyond CTV/video RTB to other programmatic advertising formats has not been assessed. The privacy-preserving measurement gap noted above is likely to grow more consequential as identifier-level targeting becomes harder to instrument directly; a follow-up study would need to establish whether the metric taxonomy in Section 4 remains diagnostically useful when built on aggregated or differentially private signals rather than per-request identifiers, and whether burn-rate alerting retains its false-positive advantages under coarser-grained data. Similarly, the framework’s applicability to non-RTB advertising formats, such as reservation-based or programmatic guaranteed deals, would need separate validation, since those formats lack the hard per-request timeout that motivates much of the latency-focused instrumentation described here.

The simulation study in Section 9 takes the first step suggested in the original submission: instrumenting the framework against a synthetic traffic dataset and reporting measured detection results for the metric segmentation of Section 4, the multi-window alerting approach of Section 5, the silence detection of Section 7, and the paired-panel revenue pattern of Section 6. That study also surfaces a qualification worth stating plainly. On a gradual, sustained latency drift, a conventional static threshold detected the regression faster than the multi-window rule (189 vs. 1,298 minutes in the simulated scenario), because the illustrative fast-burn thresholds are tuned for sharper deviations. The multi-window approach earned its advantage instead on self-correcting spikes, which it correctly ignored while the static threshold paged. Threshold and window selection therefore remains a tuning decision for adopting teams, and production validation is still needed to confirm the simulation’s findings.

11.CONCLUSION

Real-time advertising pipelines combine strict latency budgets with an immediate, direct link between technical performance and revenue, a combination generic observability practice does not fully address. Structuring metric design, alerting, and dashboarding around that link, and treating the silent absence of expected data as a failure class in its own right, gives engineering teams a fuller picture of when revenue is actually being protected or quietly lost. The framework here is offered as a starting point for teams building or maturing observability practice around advertising infrastructure, with the

expectation that the specific parameters get adapted to each pipeline's own traffic and partner mix.

Acknowledgments

Conflict of interest

The author declares no conflict of interest.

Funding

This research received no external funding.

Data availability statement

No production datasets were used in this study. The validation in Section 9 uses fully synthetic telemetry produced by the discrete-event generator described there

(fixed master seed 20260930); all generator parameters, alerting thresholds, and scenario definitions are disclosed in the paper, and the simulation code is available from the author on request. All illustrative figures in Sections 5 and 6 remain explicitly labeled hypothetical and are not derived from any dataset.

Acknowledgment of AI-assisted drafting

All cited sources, quantitative claims attributed to those sources, and the conceptual framework itself were reviewed and verified by the author prior to submission. The author takes full responsibility for the accuracy and originality of the content.

References:

- Chandra, T. D., & Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2), 225–267. DOI: 10.1145/226643.226647
- Duarte, E. P., Jr., Rodrigues, L. A., Camargo, E. T., & Turchetti, R. C. (2023). The missing piece: A distributed system-level diagnosis model for the implementation of unreliable failure detectors. *Computing*, 105(12), 2821–2845. <https://link.springer.com/article/10.1007/s00607-023-01211-8>
- Faseeha, U., Syed, H. J., Samad, F., Zehra, S., & Ahmed, H. (2025). Observability in microservices: An in-depth exploration of frameworks, challenges, and deployment paradigms. *IEEE Access*, 13, 72011–72039. <https://ieeexplore.ieee.org/document/10967524/>
- Fu, N., Cheng, G., Teng, Y., Dai, G., Yu, S., & Chen, Z. (2025). Intelligent root cause localization in microservice systems: A survey and new perspectives. *ACM Computing Surveys*, 57(12). <https://dl.acm.org/doi/10.1145/3736755>
- Jensen, S. K., Pedersen, T. B., & Thomsen, C. (2017). Time series management systems: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 29(11), 2581–2600. DOI: 10.1109/TKDE.2017.2740932
- Le, V.-H., & Zhang, H. (2022). Log-based anomaly detection with deep learning: How far are we? In *Proceedings of the 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. <https://dl.acm.org/doi/10.1145/3510003.3510155>
- Ou, W., Chen, B., Dai, X., Zhang, W., Liu, W., Tang, R., & Yu, Y. (2024). A survey on bid optimization in real-time bidding display advertising. *ACM Transactions on Knowledge Discovery from Data*, 18(3). <https://dl.acm.org/doi/10.1145/3628603>
- Raeiszadeh, M., Ebrahimzadeh, A., Saleem, A., & Glitho, R. H. (2023). Real-time anomaly detection using distributed tracing in microservice cloud applications. In *Proceedings of the 2023 IEEE 12th International Conference on Cloud Networking (CloudNet)*. <https://ieeexplore.ieee.org/document/10490038/>
- Soldani, J., & Brogi, A. (2022). Anomaly detection and failure root cause analysis in (micro)service-based cloud applications: A survey. *ACM Computing Surveys*, 55(3). <https://dl.acm.org/doi/10.1145/3501297>
- Tariq, S., Baruwat Chhetri, M., Nepal, S., & Paris, C. (2025). Alert fatigue in security operations centres: Research challenges and opportunities. *ACM Computing Surveys*, 57(9), Article 224. <https://dl.acm.org/doi/10.1145/3723158>
- Usman, M., Ferlin, S., Brunström, A., & Taheri, J. (2022). A survey on observability of distributed edge & container-based microservices. *IEEE Access*, 10, 86904–86919. <https://ieeexplore.ieee.org/document/9837035/>
- Voutsas, F., Violos, J., & Leivadeas, A. (2024). Mitigating alert fatigue in cloud monitoring systems: A machine learning perspective. *Computer Networks*, 250, Article 110543. <https://www.sciencedirect.com/science/article/pii/S138912862400375X>
- Wang, Y.-T., Ma, S.-P., Lai, Y.-J., & Liang, Y.-C. (2022). Analyzing and monitoring Kubernetes microservices based on distributed tracing and service mesh. In *Proceedings of the 2022 29th Asia-Pacific Software Engineering Conference (APSEC)*. <https://ieeexplore.ieee.org/document/10043222/>
- Yu, B., Yao, J., Fu, Q., Zhong, Z., Xie, H., Wu, Y., Ma, Y., & He, P. (2024). Deep learning or classical machine learning? An empirical study on log-based anomaly detection. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. <https://dl.acm.org/doi/10.1145/3597503.3623308>
- Zhou, X., Peng, X., Xie, T., Sun, J., Ji, C., Li, W., & Ding, D. (2021). Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering*, 47(2), 243–260. DOI: 10.1109/TSE.2018.2887384

Zhou, X., Peng, X., Xie, T., Sun, J., Li, W., Ji, C., & Ding, D. (2022). Delta debugging microservice systems with parallel optimization. *IEEE Transactions on Services Computing*, 15(1), 16–29. DOI: 10.1109/TSC.2019.2919823

Sujan Kumar Reddy Challa

Independent Researcher,
USA.

sujan.challa10@gmail.com

ORCID: 0009-0005-8705-9770
