



SvedbergOpen
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: <https://www.svedbergopen.com/>



Research Paper

Open Access

Monetization of Enterprise APIs: Metering Architecture, Quota and Billing Reconciliation, and Gateway-Level Enforcement

Rajender Reddy Muddam

Independent Researcher, India. Email: reddy.rajender89@gmail.com

Abstract

Charging for an application programming interface appears straightforward: count the calls and issue an invoice. In an enterprise setting it is not, because the count is produced by infrastructure that was built for traffic management rather than for accounting, and because a single logical interaction can present at the counting point more than once, not at all, or in a partially processed state. This paper sets out the architecture required to make enterprise interface usage billable and auditable. It argues that usage should be counted at the gateway, where every request is already authenticated and authorized, and that the gateway record should be treated as authoritative for both quota enforcement and billing so that the two cannot diverge by construction. It defines what constitutes a billable unit across event classes, including the treatment of authentication failures, server-side errors, and client retries. It develops a reconciliation process that compares gateway records against billing records before invoice generation rather than after dispute, and it addresses plan and tier transitions that must not disturb consumers mid-cycle. A particular difficulty in this setting is gateway heterogeneity: monetization capability varies substantially across commercial gateways, and a platform spanning several must reconcile differing native support behind one consistent model. The positions derive from monetization work carried out between 2022 and 2024 within a commercially deployed, gateway-agnostic interface management platform. One reconciliation defect found during validation is reported, in which client retries during transient network timeouts inflated request counts above expected billable usage. No metering accuracy percentages, call volumes, dispute rates, or revenue-impact figures were formally measured, and the paper is presented as a design contribution rather than an empirical evaluation.

Keywords— API monetization, metering architecture, billable unit, quota enforcement, billing reconciliation, API gateway, gateway-agnostic platform, usage accounting, subscription plans, rate limiting, auditability, duplicate counting, retry semantics.

I. Introduction

1.1 The apparent simplicity

Interface monetization is easy to describe and hard to implement correctly. The description is that consumers subscribe to a plan, the platform counts what they use, and an invoice follows. Every part of that sentence hides a decision that determines whether the resulting invoice can be defended.

Counting requires a definition of what is counted, and that definition has to survive requests that fail, requests that are retried, requests that are served from cache, and requests that are partially processed. Subscription requires plans that can change without disturbing consumers who are mid-cycle. The invoice requires that the number on it can be reconciled against an independent record, because a consumer who disputes a charge is entitled to an explanation, and "the system says so" is not one.

The difficulty compounds in enterprise environments, which are distributed across multiple gateways, regions, and deployment models. Usage records are produced by components whose primary purpose is traffic management rather than accounting, and analytics pipelines that are entirely adequate for observing traffic trends are not automatically adequate for producing a figure that will appear on a bill.

1.2 The setting

The positions in this paper derive from monetization work carried out between 2022 and 2024 within SAMi, a gateway-agnostic interface management platform developed at SID Global Solutions. The platform manages interfaces across multiple commercial gateways, including Apigee, MuleSoft, IBM API Connect, and Kong, through a connector-based architecture, and is designed for enterprise customers managing large interface estates and many consumer applications through a single interface. It has been commercialised, sold to multiple customers, and published on a public cloud marketplace, so the work described here was undertaken for production enterprise use rather than as a proof of concept.

The monetization contribution specifically concerned supporting monetization plans, usage metering, quota management, and integration with gateway-native monetization capability where it exists, while presenting one consistent model to the platform's users regardless of which gateway sits underneath.

The gateway-agnostic architecture of the platform itself is described in prior work [1] and is not re-presented here. This paper addresses what that architecture does not cover: how usage is metered, how quota and billing are kept consistent, and how enforcement is expressed when the underlying gateways differ in what they natively support.

1.3 Contribution and scope

This paper contributes:

1. A position on where usage should be counted, and the reasoning that makes the gateway the correct point rather than the backend.
2. A definition of the billable unit across event classes, including the cases that determine most disputes: authentication failures, server-side errors, and retries.
3. A reconciliation design that keeps quota enforcement and billing consistent by construction rather than by periodic correction, together with the process that resolves them when they nonetheless diverge.
4. A treatment of gateway heterogeneity, where native monetization support varies and a unified model must be maintained across that variation.
5. A report of one reconciliation defect found during validation, its cause, and the change made in response.

What this paper does not do. It presents no measured evaluation. Call volumes, metering accuracy against a reference count, dispute rates, and revenue affected by metering error were not formally measured. Validation was performed by comparing gateway analytics, usage logs, and expected billing outcomes across usage scenarios to confirm that metering and quota enforcement behaved consistently, which establishes behavioural correctness under test rather than accuracy in production. Section 11 states the evidence position in full, and Section 13 records the limitations that follow from it.

2. Background and Related Work

Interface management and monetization in practice. Monetization is treated in the practitioner literature as a capability of interface management platforms [2], typically covering plan definition, key issuance, quota configuration, and reporting. That treatment establishes the feature surface. It generally assumes a single gateway and treats the usage record as a given rather than as something whose construction is itself the difficult part.

Metering and billing as a specified discipline. Standards work on cloud service metering elements and billing modes [3] establishes that metering deserves definitional rigour, distinguishing the measured element from the billing mode applied to it. This framing is directly useful, because most monetization disputes are traceable to an ambiguity in one or the other, and it is the framing this paper adopts when defining the billable unit in Section 4.

Gateway architecture. Gateway deployment models, including self-hosted arrangements where the gateway runs inside the customer's own boundary [4], bear on metering because they determine whether usage records leave the customer environment promptly, in batches, or on a delay. A design assuming immediate availability of usage data will behave differently under a self-hosted deployment, and Section 5 treats this as a first-order concern rather than an operational detail.

Reconciliation rather than distributed transaction. The argument that large-scale systems should abandon distributed transactions in favour of activities that are explicitly reconciled [5] describes precisely the position a monetization platform occupies. Usage is produced in one place and billed in another; the two cannot be updated atomically, and the correct response is not to seek atomicity but to design the reconciliation deliberately. Section 6 develops that position.

Prior work on the platform. The gateway-agnostic management framework within which this monetization capability sits is described separately [1]. That work addresses decoupling interface design, governance, and runtime policy from specific gateway implementations. It does not address metering, quota reconciliation, or billing, which are the subject of this paper. The relationship is that the present work is a capability built on the architecture described there, and the reader interested in the connector model and policy abstraction should refer to it.

The gap this paper addresses is the absence, in both the standards and the practitioner literature, of a treatment of metering that takes gateway heterogeneity and record reconciliation as central rather than incidental.

3. Why Accurate Metering Is Difficult

Four properties of enterprise environments make usage accounting harder than the product description suggests.

Distribution. Deployments span multiple gateways, regions, and deployment models. Each produces usage records in its own format, on its own schedule, with its own definition of what constitutes a recorded event. A platform presenting one number to the customer is aggregating across sources that do not agree on terms.

Request ambiguity. A single logical interaction may be retried by the client, served from cache, routed through several services, or partially processed. Each of these produces a different relationship between what the consumer intended and what the counting point observed, and only some of them should result in a charge.

Model variety. Organisations charge in different ways: per request, per transaction, or by subscription tier. The same underlying event stream must support all of these, which means the metering layer cannot bake in a single notion of what a unit is.

Deployment model changes the guarantees available. A gateway operating as a managed cloud service, one self-hosted inside a customer's network boundary, and one running at the edge across regions offer materially different assurances about when a usage record becomes visible and whether it can be lost in transit. A metering design that assumes the strongest of these and is deployed against the weakest will under-count silently, and silence is the operative word: nothing in the pipeline reports an absence.

Temporal skew. Analytics data arrives with delay, and the delay is variable. A usage figure read at the moment a billing period closes is not the same figure that will be available an hour later, and a design that ignores this will produce invoices that disagree with the records they were derived from.

Caching breaks the correspondence between request and work. A response served from cache consumed almost no backend resource but was, from the consumer's position, a delivered service. Whether it is billable is a commercial question rather than a technical one, and a metering layer that does not distinguish cached from origin-served responses has removed the ability to answer it either way.

Partial processing has no natural unit. A request routed through several services and failing at the third has consumed real capacity and delivered nothing. Counting it as a full unit overcharges; discarding it undercharges and, worse, creates an incentive structure in which a provider benefits from the consumer's failures being invisible.

The consequence is that consistent tracking, duplicate avoidance, delayed-event handling, and reconciliation between gateway metrics and billing systems all require deliberate design. None emerges from correctly configuring a gateway.

It is worth naming the underlying reason these problems are systematic rather than incidental. Gateways were built to manage traffic, and traffic management is tolerant of exactly the imprecision that billing is not. A rate limiter that counts a request twice under load has behaved conservatively. A dashboard that loses a fraction of a percent of events still shows the right trend. Neither property is a defect in the context the component was designed for, and both become defects the moment the same numbers are used to charge someone. A monetization platform is therefore not configuring a gateway to do accounting; it is building an accounting layer on top of a component that was never asked to be one.

4. Where Usage Is Counted and What Counts

4.1 The gateway is the counting point

Usage is counted at the gateway. The reasoning is that the gateway is the first point at which every request is authenticated, authorized, and processed before reaching any backend service, which gives it three properties no backend possesses.

It sees every request, including those that never reach a backend because they were rejected at the perimeter. A backend counter is blind to precisely the traffic that consumes capacity without producing work.

It sees requests uniformly, regardless of which backend eventually serves them. Counting in backends means maintaining consistent counting logic across every service, and that consistency degrades the moment teams own their services independently.

It is already an enforcement point. Since quota enforcement must happen where requests can be rejected, and that is the gateway, counting there means enforcement and billing draw on the same observations rather than on two independently maintained ones.

4.2 Why not the backend

The alternative, counting where the work is performed, has surface appeal: the backend knows whether a request produced value, which the gateway cannot always tell. Three problems rule it out.

Counting in backends means every service must implement counting consistently, and consistency across independently owned services degrades rather than holds. A team optimising its own service has no reason to preserve a counting behaviour whose only consumer is a billing system it does not operate.

Backends are blind to rejected traffic. A consumer sending large volumes of unauthorised requests consumes gateway capacity and produces no backend record at all, which means the counting point cannot observe the traffic most likely to require a commercial conversation.

And enforcement would be separated from counting. Quota decisions must be made where a request can still be refused, which is the gateway. If billing counted elsewhere the two would derive from different observations, and the consistency argument of Section 6 would be unavailable.

The residual difficulty, that the gateway cannot always tell whether a request produced value, is real, and it is addressed by the event classification in Table 1 rather than by moving the counting point.

4.3 Defining the billable unit

A billable unit represents a successful request or transaction, with the specific definition depending on the customer's plan, which may charge per call, per transaction, or by subscription tier. The definitional work is in the edge cases, and Table 1 states the position for each.

Table 1. Event classes and their billing treatment.

Event class	Billed	Reasoning
Successful request	Yes	The consumer received the service contracted for
Authentication failure	No	The consumer was never admitted; charging for rejection is indefensible
Authorization failure	No	Same reasoning as authentication failure
Server-side error after admission	Generally yes	The gateway processed the request and consumed capacity, though plan terms may vary this
Client retry of a prior request	Once	Deduplicated by request identifier; see Section 4.4
Cached response	Depends on plan	The consumer received a result; whether that constitutes billable consumption is a commercial decision, not a technical one

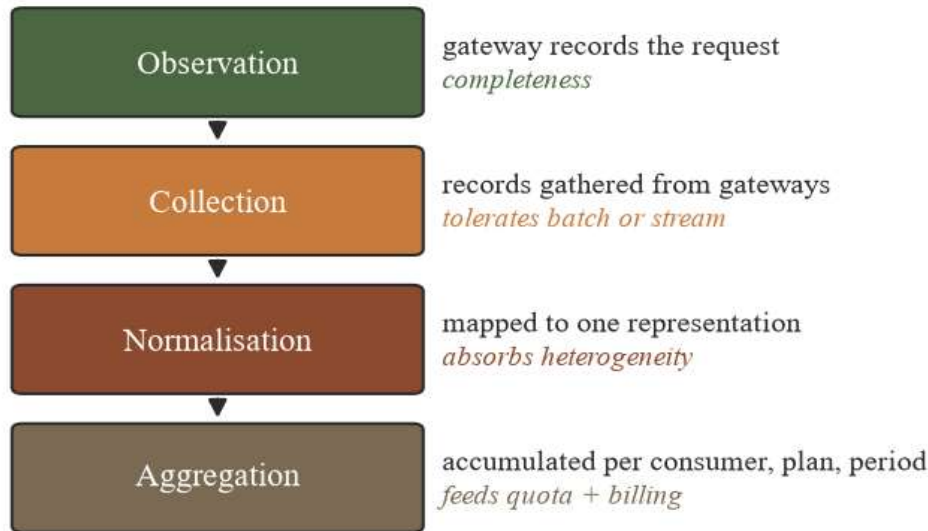
The server-side error row is the one that most often requires explicit agreement with the customer. The technical position is that the request was admitted and consumed gateway capacity. The commercial position a customer may reasonably take is that they did not receive what they paid for. Because both are defensible, the resolution belongs in the plan terms rather than in the metering implementation, and the metering layer's obligation is to make the category visible so that either treatment can be applied.

4.4 Retries

Retries are handled using request identifiers or idempotency mechanisms so that repeated delivery of the same logical request does not produce repeated charges. This matters commercially as well as technically: a consumer whose client library retries aggressively on timeout would otherwise be billed for its own error handling, which is the least defensible charge a platform can issue. Section 12 reports a case where exactly this occurred during validation.

5. Metering Architecture

The metering path has four stages, shown in Figure 1, and the properties that matter differ at each.



Every record carries a request identifier. Every stage is idempotent with respect to it, so redelivery never becomes extra usage.

Figure 1. The metering path. Completeness governs observation because a record missed there cannot be recovered. Every later stage is idempotent with respect to the request identifier, so redelivery never becomes additional usage.

Observation. The gateway records the request, including its identity, its consumer, its plan association, its outcome, and a timestamp. The critical property here is completeness rather than speed. A record missed at observation cannot be recovered downstream, whereas a record delayed can.

Collection. Records are gathered from gateways into the platform. This is where deployment model asserts itself: a self-hosted gateway inside a customer boundary [4] may deliver records in batches on a schedule rather than continuously, so the collection stage must tolerate both arrival patterns without changing the resulting usage figure.

The stages are worth separating explicitly because they fail differently and are usually built as one component, which is why their failure modes get conflated. Observation failures are unrecoverable. Collection failures are recoverable if the source retains the record. Normalisation failures are recoverable at any time because the source record persists. Aggregation failures are recoverable by recomputation. Knowing which class a defect belongs to determines whether the response is to fix forward or to reprocess, and a pipeline built as a single stage forces the most expensive answer every time.

Normalisation. Records from different gateway products are mapped to one internal representation. This is where gateway heterogeneity is absorbed, and it is the stage most specific to a gateway-agnostic platform. What one gateway reports as a single field may require derivation on another, and the normalised record must carry enough provenance that a figure can be traced back to the gateway record that produced it.

Aggregation. Normalised records are accumulated against consumers, plans, and periods, producing the figures used for both quota enforcement and billing.

Two design commitments run through all four stages. Every record carries a request identifier, which is what makes deduplication possible at any later stage rather than only at the point of observation. And every stage is idempotent with respect to that identifier, so that a record delivered twice by a retrying collection process does not become two units of usage. Without this second property the collection layer becomes a source of the exact defect the metering layer exists to prevent.

6. Quota and Billing Reconciliation

6.1 Consistency by shared source

Quota enforcement and billing stay consistent because both derive from the same gateway usage metrics. This is a structural choice rather than a process one. Two systems reading one source can drift only through processing differences, which are findable. Two systems reading separate sources will diverge for reasons that are not findable, because there is no single record against which to check either.

The corollary is that audited gateway records are treated as the authoritative source. When any two records disagree, the gateway record is the one that governs, and the others are corrected toward it. Establishing this in advance rather than at dispute time is what makes disputes resolvable.

The choice also determines who can answer a question. When one record governs, any figure can be traced to it by anyone with access, and the answer does not depend on which system a given analyst happens to trust. When several records coexist without a stated precedence, answering a customer question becomes an exercise in judgement, and different people will answer the same question differently. That inconsistency is more damaging commercially than a small inaccuracy would be, because a customer can accept that a number was slightly wrong far more readily than that the provider does not appear to know which of its numbers is real.

6.2 Reconciliation before invoicing

Usage data is collected continuously and reconciled periodically against analytics and billing records. Where differences arise, whether from delayed events, retries, or synchronisation issues, the reconciliation process compares gateway logs against billing data and corrects the usage figure before the invoice is generated.

The sequencing is the point. Reconciling before invoicing means discrepancies are resolved internally, and the customer sees one correct figure. Reconciling after invoicing means the customer discovers the discrepancy, which converts an internal data problem into a commercial conversation and a credit note. The same reconciliation logic serves both, but its position relative to invoice generation determines whether it prevents disputes or merely explains them.

6.3 When the two disagree at period end

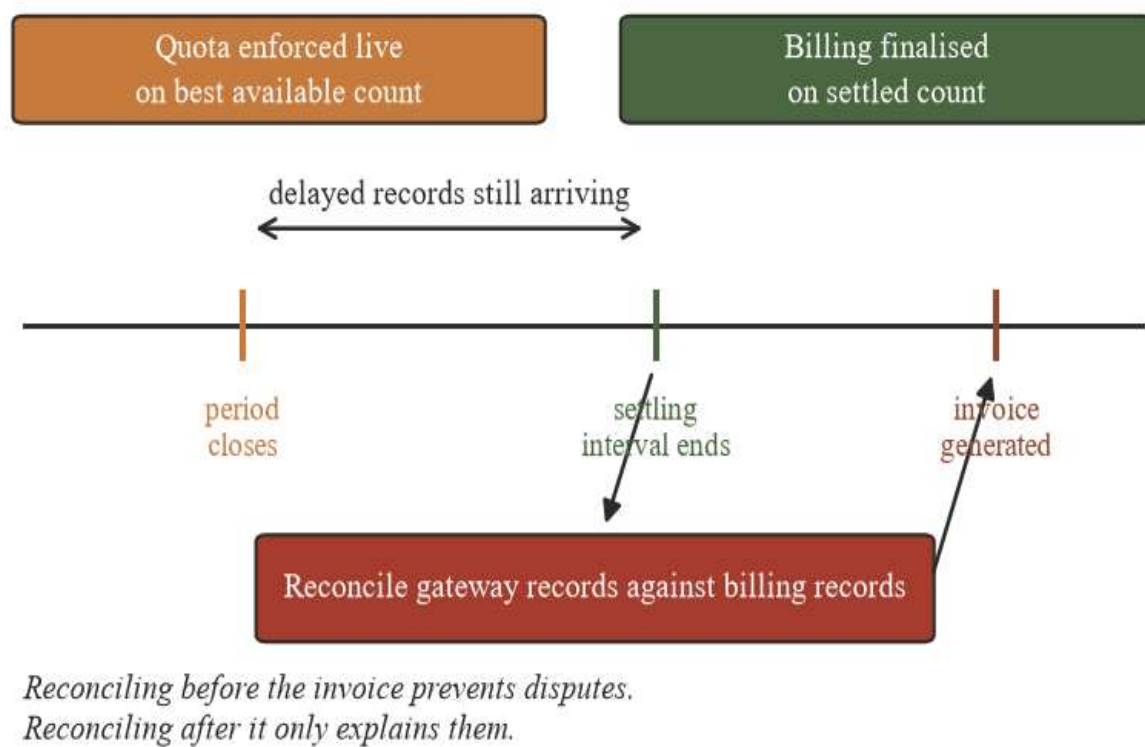


Figure 3. Timing of enforcement, settling, and invoicing. Quota decisions use the best figure available in the moment. Billing waits for the settled figure, and reconciliation runs against it before the invoice is generated.

Disagreement at period close is expected rather than exceptional, because delayed analytics guarantee that the figure available at the moment of closing is incomplete. The design accommodates this rather than treating it as an error condition: the period is closed for enforcement purposes at the boundary, while the billing figure is finalised only after a settling interval sufficient for delayed records to arrive, with reconciliation run against the settled data.

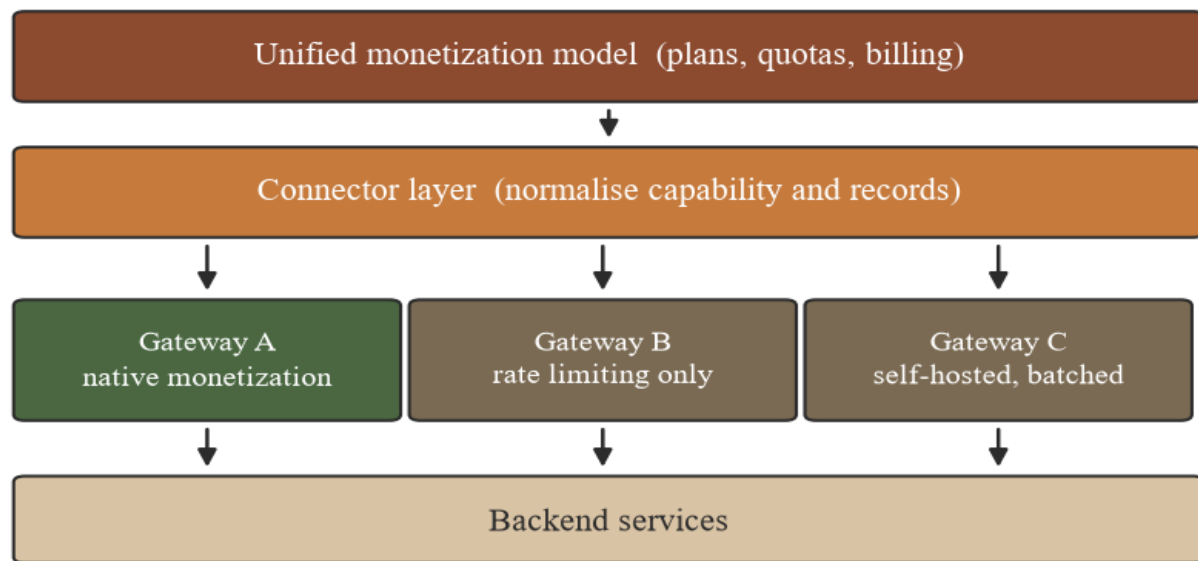
This introduces a deliberate asymmetry. Quota decisions must be made in real time and therefore use the best available figure, which may later prove slightly low. Billing decisions can wait and therefore use the settled figure. A consumer may consequently be permitted a request that a fully settled count would have blocked. That

is the correct trade, because the alternative is to delay enforcement until data settles, which would make quotas unenforceable in the moment they need enforcing.

Enforcement happens at the gateway because that is where a request can be refused before it consumes backend capacity. The complication in a gateway-agnostic platform [1] is that gateways differ substantially in what they natively provide.

Some gateways offer native monetization: plan definitions, quota enforcement, and usage reporting as first-class product features. Others offer rate limiting without a commercial model attached, leaving plan semantics to be expressed by whatever sits above. A platform spanning both must present one consistent model to its users while mapping that model onto substantially different substrates.

Figure 2 shows how the connector layer sits between one commercial model and several substrates, and Table 2 sets out the variation it must absorb



The platform owns the commercial model and delegates enforcement, so an invoice does not depend on which gateway sits underneath.

Figure 2. Absorbing gateway heterogeneity. The platform owns plans, quotas, and billing; the connector layer maps them onto whatever each gateway natively provides.

7. Gateway-Level Enforcement Across Heterogeneous Gateways

Table 2. Capability variation the platform must reconcile.

Capability	Where natively available	Where not	Platform response
Plan and product definition	Gateways with native monetization	Gateways offering only traffic policy	Define in the platform, project onto gateway constructs
Quota enforcement	Broadly available as rate limiting	Semantics differ in window and granularity	Normalise to one quota model, configure per gateway
Usage reporting granularity	Varies by product	Varies by product	Normalise at the metering layer, retain provenance
Billing integration	Rare natively	Common	Perform in the platform in all cases
Analytics delivery latency	Varies with deployment model	Varies with deployment model	Settling interval before billing finalisation

The design position is that the platform owns the commercial model and delegates enforcement, using native capability where it exists and supplying the equivalent where it does not. The alternative, adopting whichever model the underlying gateway provides, would produce different behaviour for the same plan on different

gateways, which defeats the purpose of a unified platform and would make a customer's invoice depend on an infrastructure choice they may regard as incidental.

8. Plan and Tier Evolution

Plans change while consumers are using them, and a change must not disrupt service or corrupt the current period's accounting. These are two distinct obligations and they fail in different ways: a disruption is immediately visible to the consumer and generates a support contact, whereas a corrupted accounting period is invisible until the invoice arrives, by which point the underlying records may already have been aggregated. The second is the more damaging of the two precisely because it surfaces late.

The approach is versioned subscription policy that preserves existing entitlements until the current billing cycle ends. When a customer changes plan, the new configuration takes effect either immediately with prorated billing or at the start of the next cycle, according to the business rules that apply. Existing keys, quotas, and access permissions remain valid throughout the transition, so no consumer experiences an interruption caused by a commercial change.

Figure 5 shows the timing and Table 3 states the semantics.

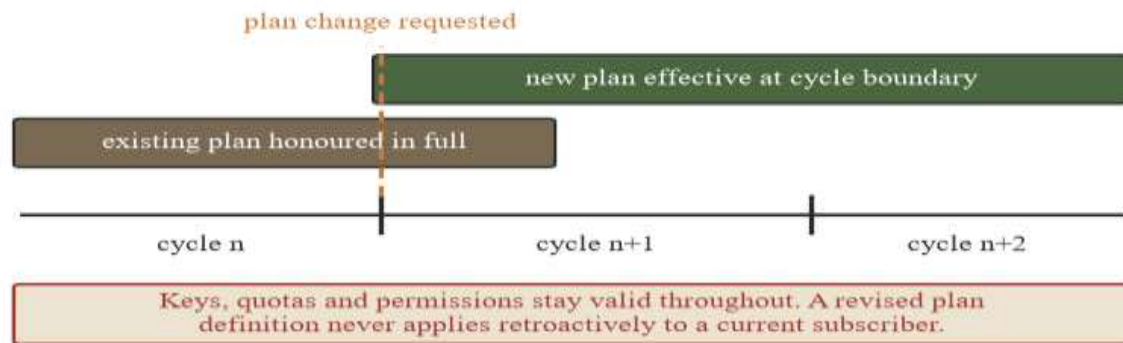


Figure 5. Plan transition across a cycle boundary. Credentials and quotas remain valid throughout, and a revised plan definition never applies retroactively to a consumer already subscribed under the previous one.

Table 3. Plan transition semantics.

Transition	Timing	Entitlement during transition	Accounting
Upgrade, immediate	On confirmation	Existing credentials remain valid, quota raised	Prorated across the cycle
Upgrade, scheduled	Next cycle start	Existing plan continues in full	Clean cycle boundary, no proration
Downgrade	Next cycle start	Existing plan continues in full	Clean cycle boundary
Plan definition revised	Existing subscribers on prior version	Prior version honoured to cycle end	Prior version's terms apply

The last row carries the important commitment. Revising a plan definition does not retroactively alter what current subscribers agreed to, because a change to plan terms that took effect mid-cycle would mean a consumer was charged under terms they never accepted. Versioning the subscription policy is what makes this a structural guarantee rather than a matter of operational care.

9. Failure Modes and Detection

Three failure classes recur in usage accounting.

Duplicate counting arises principally from client retries, where one logical interaction is observed as several. It is detected through request identifiers and through reconciliation comparing expected against actual usage.

Lost or delayed usage events arise from gateway or network interruption. The distinction between lost and delayed matters and is not immediately observable, since both present initially as a shortfall against

expectation. Only the passage of the settling interval separates them, which is a further reason for the interval described in Section 6.3.

The distinction between the first two matters operationally as well as conceptually. Duplicate counting is detectable from within a single record set, because the duplicates are present and identifiable by their shared request identifier. Lost events are not detectable from within the record set at all, because what is missing leaves no trace there; they are detectable only by comparison against an expectation formed elsewhere. This asymmetry means the two require different detection machinery, and a platform that has built deduplication well is not thereby protected against loss.

Boundary effects arise where usage is recorded near a billing-cycle close or a quota reset. A record timestamped just before a boundary but delivered just after must be attributed by its timestamp rather than its arrival, and timestamp validation exists to enforce that.

A fourth class deserves separate mention because it is the hardest to detect. Silent under-counting, where usage events are never generated rather than generated and lost, produces no anomaly in any downstream comparison, because every system agrees on a figure that is uniformly too low. Reconciliation between gateway records and billing records cannot surface it, since both derive from the same absent observations. Only comparison against an independent count, of the kind Section 15 proposes, would reveal it. This is the strongest argument for obtaining such a count, and it is a different argument from wanting an accuracy figure for its own sake.

Detection rests on gateway analytics, audit logs, request identifiers, timestamp validation, and periodic reconciliation. Monitoring dashboards and automated alerting surface unusual usage patterns, and reconciliation compares expected against actual before invoices are generated or quotas finalised. Table 4 pairs each failure mode with its detection route.

Table 4. Failure modes and detection.

Failure mode	Typical cause	Detection	Point of correction
Duplicate counting	Client retry, collection retry	Request identifier deduplication, reconciliation variance	Before invoice generation
Lost usage events	Gateway or network interruption	Reconciliation shortfall persisting past settling interval	Manual investigation against gateway logs
Delayed usage events	Analytics pipeline latency, batched delivery	Shortfall resolving within settling interval	Automatic on arrival
Boundary misattribution	Record crossing a cycle or reset boundary	Timestamp validation against arrival time	At aggregation
Plan misapplication	Transition timing error	Reconciliation against subscription policy version	Before invoice generation

9.1 Why detection has to be scheduled rather than triggered

Most of the failure modes above produce no alarm at the moment they occur. A lost usage event raises nothing, because the component that would have raised it is the one that failed. A boundary misattribution raises nothing, because every record involved is individually valid. A duplicate raises nothing until something compares records to each other.

This means detection cannot be event-driven in the way operational alerting usually is. It has to be a scheduled comparison that runs whether or not anything appears wrong, because the class of defect that matters here is defined by the absence of a signal rather than by its presence. Reconciliation is therefore not a remediation step invoked when a discrepancy is suspected; it is the mechanism by which discrepancies become suspectable at all.

Table 5 states which failure modes are detectable from within a single record set and which require an external comparison, because that distinction determines what machinery is needed.

Table 5. Detectability of each failure mode.

Failure mode	Visible within one record set	Requires external comparison
Duplicate counting	Yes, duplicates share a request identifier	No

Boundary misattribution	Yes, timestamp against arrival time	No
Delayed usage events	Partly, resolves on its own	Only to confirm resolution
Lost usage events	No, absence leaves no trace	Yes, against an expected figure
Silent under-counting at source	No	Yes, against an independent count

The practical consequence is that reconciliation must run on every period regardless of whether the period looked normal, and its output should be recorded even when it finds nothing. A reconciliation that produces no artifact when it passes cannot later be shown to have run, which matters when a customer asks what checking was done before their invoice was issued.

10. Auditability of the Usage Record

An invoice is a claim about what happened. In an enterprise relationship it must be a claim the provider can substantiate, and that imposes requirements on the usage record beyond being correct.

Provenance must survive normalisation. Section 5 describes mapping records from different gateway products into one representation. That mapping is a transformation, and a transformed record that cannot be traced back to the gateway record it came from is not audit evidence. Each normalised record therefore retains the identity of its source record and the gateway that produced it, so any figure on an invoice can be decomposed into the observations behind it.

Corrections must be additive. When reconciliation adjusts a usage figure, the adjustment is recorded rather than the original being overwritten. A record that silently becomes correct cannot be distinguished from one that was always correct, and showing what was adjusted and why is exactly what a customer asking about a correction needs to see.

Retention is set by the dispute window, not by storage cost. Usage records must remain available for as long as a charge derived from them can be questioned, which is generally longer than an operational analytics retention policy would choose. This deserves a deliberate decision, because analytics retention is usually configured with dashboards in mind and its default will be too short.

Reports and invoices must derive from the same records. A usage report generated from analytics while the invoice is generated from the billing system reintroduces precisely the divergence Section 6 eliminates, one layer further out and where the customer can see it. The report a customer receives should be a view of the record the invoice was computed from.

Together these make the usage record an accounting artifact rather than a telemetry byproduct, and that reclassification is the point. Telemetry is allowed to be lossy, approximate, and aggressively expired, and all three are reasonable for observing traffic. None is acceptable for a record supporting a financial claim.

11. Dispute Investigation and Resolution

A monetization platform is judged less by whether it bills correctly than by whether it can demonstrate that it billed correctly when a customer says otherwise. The two are different capabilities, and the second is the one that determines whether a commercial relationship survives a disagreement.

11.1 The investigation path

A dispute is investigated by reviewing gateway analytics, access logs, usage reports, and billing records for the disputed period. The objective is narrow and should be stated as such: establish whether the recorded consumption matches what the consumer actually did. Everything else, including whether the charge was fair under the plan, is a commercial question that can only be addressed once the factual one is settled.

This is where the design decisions of Sections 4 and 6 pay for themselves. Because usage is counted at one point and the audited gateway record is authoritative, an investigation has a defined place to look and a defined record to trust. A platform that counted in several places would begin every dispute by deciding which of its own numbers to believe, and that conversation is not one a customer should ever observe.

11.2 What disputes are actually about

Table 6 sets out the causes that recur, and the pattern in them is worth drawing out.

Table 6. Recurring dispute causes and where they are resolved.

Cause	Nature	Where it is resolved
Misunderstanding of the pricing plan	Commercial, not technical	Plan documentation and usage report clarity

Duplicate requests from client retries	Technical, on the consumer side	Deduplication before billing, as in Section 12
Delayed synchronisation of usage data	Technical, on the platform side	Settling interval before invoice finalisation
Divergence between consumer application logs and gateway metrics	Definitional	Explaining what the gateway counts and why

Only one of these four is a defect in the platform's accounting. One is a commercial misunderstanding, one originates in the consumer's own client behaviour, and one is a disagreement about what is being counted rather than about the count. This distribution has a design implication that is easy to miss: most dispute-reduction effort is better spent on making the usage report legible than on making the counter more accurate, because most disputes are not about accuracy.

The fourth row deserves particular attention. A consumer comparing its own application logs against the platform's usage report is comparing two different measurements, taken at different points, of things that are not quite the same. The consumer counts what its application intended to send. The gateway counts what arrived. Retries, client-side timeouts, and connection failures separate the two, and neither number is wrong. Anticipating this in the usage report, by exposing the categories that account for the difference rather than only the total, converts a dispute into an explanation.

11.3 Resolution and what it produces

Resolution involves reconciling the records, identifying the root cause, and correcting the billing where correction is warranted. Detailed audit logs and transparent usage reports are what make this tractable, and they are also what make it fast, which matters because the cost of a dispute to the relationship is closer to a function of how long it takes to resolve than of how large the disputed amount was.

Each resolved dispute also produces information the platform should use. A dispute traced to a plan misunderstanding indicates the plan terms or the report presentation need work. A dispute traced to retries indicates the deduplication boundary needs examination. Treating the dispute corpus as a defect-detection channel rather than as a customer-service workload is what turns it from a cost into a signal, and Section 14 identifies analysing that corpus as the most useful measurement this work has not yet performed.

11.4 What the dispute process implies for product design

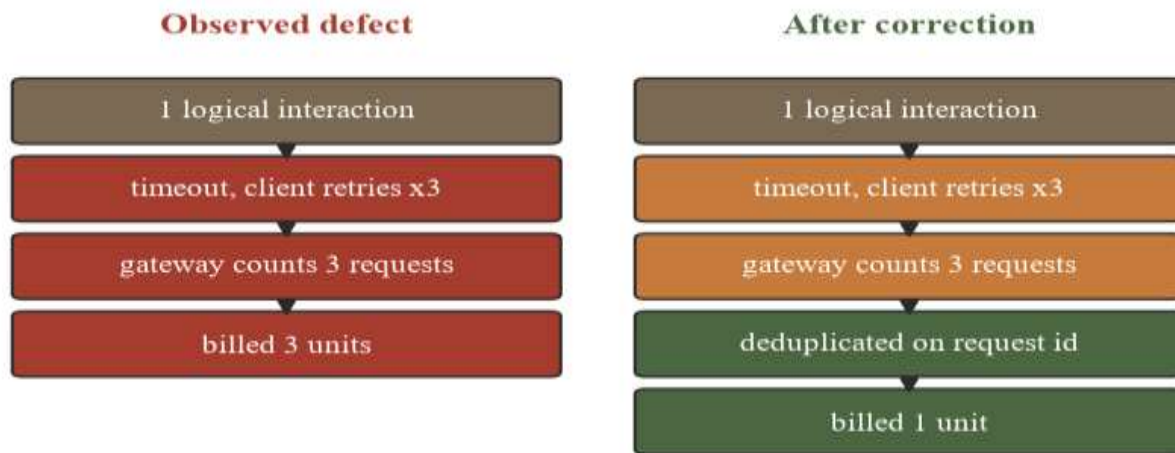
Two design obligations follow from Section 11.2 that are easy to treat as reporting cosmetics rather than as architecture.

The usage report must expose its own composition. A total is not defensible on its own, because a customer disputing it cannot engage with a single number. Presenting the total alongside the categories that make it up, successful requests, rejected requests excluded from billing, retries collapsed by deduplication, and any reconciliation adjustments applied, converts the report from an assertion into an argument the customer can follow. This is a requirement on the aggregation stage of Section 5, not on the reporting layer, because a stage that discards the categories cannot have them presented later.

The plan must be legible in the same terms the report uses. A substantial share of disputes originates in a mismatch between what a customer understood they were buying and what the platform counts, and that mismatch is often introduced by describing the plan in commercial language while reporting usage in technical language. If a plan is sold as a number of transactions and the report shows a number of requests, the two must either be the same quantity or the report must show the relationship between them. Otherwise the customer is asked to reconcile the platform's vocabulary on the platform's behalf, and they will reasonably decline.

12. A Reconciliation Defect Found During Validation

During validation of the monetization workflow, a discrepancy was identified in which repeated client retries during transient network timeouts produced gateway request counts higher than the customer's expected billable usage.



Nothing malfunctioned. Raw request arrivals and billable interactions are different quantities.

Figure 4. The retry-inflated counting defect and its correction. The gateway counted correctly in both cases. What changed is the step between counting and billing that collapses repeated deliveries of one interaction into a single billable unit.

Investigation using gateway analytics and request logs established the cause: some retries were being counted as separate requests rather than recognised as repeated delivery of one logical interaction. The failure was not in the gateway's counting, which accurately recorded what it received, but in the absence of a step between counting and billing that would collapse repeated deliveries of the same request into one billable unit.

The response was to strengthen reconciliation by incorporating unique request identifiers and additional validation checks

before usage reports are finalised. This reduced the risk of duplicate billing and improved the transparency of usage reporting, since a consumer's usage report now reflects logical interactions rather than raw request arrivals.

Two observations are worth drawing out, and both informed the architecture in Section 5.

The defect was in the interpretation of correct data. Every component behaved as specified, and the number was still wrong for billing purposes, which is characteristic of metering defects generally. The gateway's job is to count requests; deciding which requests constitute billable usage is a different job, and a system that conflates them will produce a defensible traffic figure and an indefensible invoice.

The failure mode is asymmetric in its consequences. Over-counting produces an overcharge, which the customer discovers and disputes, damaging trust in the platform's accounting. Under-counting produces lost revenue, which the provider discovers eventually if at all. Since over-counting is the more damaging failure, the deduplication design deliberately prefers to under-count in ambiguous cases rather than to charge for an interaction whose distinctness cannot be established.

This finding arose in validation rather than in production, and it is reported as such. No production billing dispute traceable to a metering defect is claimed.

13. Evidence and Its Limits

The evidence position must be stated precisely, because monetization is a domain where unsupported accuracy claims are both common and consequential.

What is established. The platform is commercially deployed, sold to multiple customers, and distributed through a public cloud marketplace, so the monetization capability described here operates in real enterprise use rather than as a prototype. The design positions were exercised during validation by comparing gateway analytics, usage logs, and expected billing outcomes across usage scenarios, confirming that metering and quota enforcement behaved consistently under those scenarios. The reconciliation defect in Section 12 was found, diagnosed, and corrected.

What is not established. Call volumes, metering accuracy measured against an independent reference count, dispute rates, and revenue affected by metering error were not formally measured. Table 7 records the status of each claim class.

Table 7. Evidence status by claim class.

Claim class	Status
-------------	--------

Platform commercially deployed and sold	Established. Public marketplace listing
Metering and quota behave consistently across usage scenarios	Validated under test scenarios, not measured in production
Retry-inflated counting defect and its correction	Observed during validation, diagnosed, corrected
Gateway as authoritative source	Design position
Billable unit definitions per event class	Design position, subject to plan terms
Reconciliation before invoicing	Design position and practice
Plan transition semantics	Design position and practice
Metering accuracy against a reference count	Not measured
Call volumes, dispute rates, revenue impact	Not measured
Enterprise scale capability	Design target. No documented deployment figures

The distinction between the second and eighth rows is the one that matters most. Confirming that metering behaves consistently across scenarios establishes that the system does what it was designed to do under conditions the designers anticipated. Measuring accuracy against an independent reference count would establish that what it was designed to do is correct. Only the first was performed, and the difference between them is the gap between validation and evaluation.

14. Limitations

No measured accuracy. The central quantity in any metering system, the divergence between the count produced and the count that ought to have been produced, was not measured against an independent reference. This is the most significant limitation and it bounds every other claim in the paper.

The commercial and technical positions on error billing were not tested against customers. Section 4 states that server-side errors are generally billed because the gateway processed the request, while acknowledging the customer may reasonably disagree. That framing is presented as a design position resolved through plan terms. Whether customers in practice accept it, negotiate it, or dispute it was not studied, and it is the sort of question where the technical argument and the commercial reality can diverge considerably.

Validation scenarios were designed by the implementers. Scenario-based validation confirms behaviour under anticipated conditions. It is weak evidence about unanticipated ones, and metering defects characteristically arise from conditions nobody modelled, as the retry case in Section 12 illustrates.

Settling interval is unquantified. Section 6.3 requires an interval sufficient for delayed records to arrive, without stating one. The correct value depends on the analytics latency of each gateway product and deployment model, and that latency was not characterised.

The platform's own counting has not been differentially tested against gateway-native counting. Where a gateway provides native monetization, two independent counts of the same traffic exist, and any systematic divergence between them would be informative. That comparison has not been performed systematically, which means the platform's normalisation layer has not been checked against the substrate it normalises.

Gateway coverage is uneven. The monetization work concentrated on gateways with native monetization capability. The position that the platform supplies equivalent capability where it is absent is architecturally stated in Section 7 but has been exercised less thoroughly, and the reader should treat the claim of uniform behaviour across gateways as a design intent that is more established for some substrates than others.

No production dispute data. Section 5 of the questionnaire underlying this work describes dispute investigation as a process. No corpus of resolved disputes was analysed, so the assertion about which causes predominate reflects operational expectation rather than measured distribution.

Scale figures are absent. The platform is designed for estates comprising large numbers of interfaces and consumer applications. No documented figures for deployed estate size are presented, so its behaviour at the upper end of its intended range is not evidenced here.

15. Future Work

Measure metering accuracy against an independent reference. Instrumenting a controlled interface with a known call count and comparing the metered figure against it would produce the accuracy number this work lacks. It is the single most valuable missing measurement and is straightforward to obtain.

Characterise analytics latency per gateway and deployment model. This would convert the settling interval from a judgement into a derived parameter, and would allow it to vary by substrate rather than being set conservatively for the slowest.

Test the platform's own metering against gateway-native monetization where both exist. On gateways that provide native monetization, the platform and the gateway are independently counting the same traffic. Comparing the two produces a cross-check that requires no additional instrumentation, and any systematic divergence between them is informative regardless of which is closer to truth. This is the cheapest verification available and it has not been performed systematically.

Analyse a dispute corpus. Once a body of resolved disputes exists, classifying them by root cause would replace the expectation in Section 9 with a measured distribution, and would indicate where reconciliation effort produces the greatest return.

Instrument the settling interval. Recording how usage figures converge after a period closes, per gateway and deployment model, would show whether the interval currently applied is generous, tight, or wrong, and would allow it to be set per substrate rather than conservatively for the slowest.

Define the settling interval per gateway rather than globally. A single conservative interval delays invoicing for every substrate to accommodate the slowest, which has a real commercial cost in days-sales-outstanding across a customer base. Deriving it per gateway from measured latency would recover most of that delay without weakening the guarantee.

Extend deduplication semantics. The correction in Section 12 deduplicates on request identifier. A logical interaction that a consumer regards as one unit may nonetheless span several distinct requests, and a richer notion of a billable transaction would allow plans expressed in business terms rather than in request counts. This is the direction that would most improve the fit between what customers believe they are buying and what the platform counts.

16. Conclusion

The wider observation is that monetization sits at a junction between two disciplines with incompatible tolerances. Traffic engineering is properly tolerant of approximation, because a rate limiter that occasionally over-counts has erred safely and a dashboard that drops a fraction of a percent still shows the truth. Accounting is not tolerant of approximation at all, because every figure must be defensible to someone who did not produce it. Building the second on top of the first is the actual engineering problem, and most of the design positions in this paper are consequences of taking that mismatch seriously rather than assuming that a sufficiently good gateway configuration resolves it.

Interface monetization fails in a characteristic way. The infrastructure counts correctly, the pipeline delivers faithfully, the invoice is generated from the delivered figure, and the number is still wrong, because nobody decided what a billable unit was with enough precision to survive retries, failures, and delayed records.

This paper has argued for counting at the gateway, because that is where every request is already seen and where enforcement must occur in any case; for treating the audited gateway record as authoritative for both quota and billing, so that the two cannot diverge by construction; for defining the billable unit explicitly across event classes, including the awkward ones where the technical and commercial positions differ; for reconciling before invoicing rather than after dispute; and for absorbing gateway heterogeneity in the platform so that a customer's invoice does not depend on which gateway happens to sit underneath.

The defect reported in Section 12 illustrates the general shape of the problem. Client retries during transient timeouts inflated request counts, not because anything malfunctioned, but because raw request arrivals and billable interactions are different quantities and the system had not yet been told so.

What this paper does not offer is a measured accuracy figure, and that absence is the honest boundary of its contribution. The design positions were validated against scenarios rather than evaluated against a reference count. Obtaining that count is the natural next step and would convert the central claim of this work from a defensible design argument into an evidenced one.

References

- [1] R. R. Muddam, "A Framework for Gateway-Agnostic API Management in Distributed Enterprise Systems," *International Journal of Emerging Trends in Computer Science and Information Technology*, vol. 7, no. 2, 2026. doi: 10.63282/3050-9246.ijetcsit-v7i2p107.
- [2] B. De, "API Monetization," in *API Management: An Architect's Guide to Developing and Managing APIs for Your Organization*. Berkeley, CA, USA: Apress, 2017, pp. 143-152. doi: 10.1007/978-1-4842-1305-6_8.
- [3] BSI, Information Technology. Cloud Computing. Cloud Service Metering Elements and Billing Modes, BSI British Standards. doi: 10.3403/30395986.
- [4] S. Malvik, "Self-hosted API Gateway," in *Mastering Azure API Management*. Berkeley, CA, USA: Apress, 2022, pp. 187-195. doi: 10.1007/978-1-4842-8011-9_14.
- [5] P. Helland, "Life Beyond Distributed Transactions," *ACM Queue*, vol. 14, no. 5, pp. 69-98, 2016. doi: 10.1145/3012426.3025012.