Cowboy Queues: CBQS

Technical Whitepaper

Charles DePue, Cowboy Foundation

November 5, 2026 — Draft v0.2.1

Abstract

This document describes the **Cowboy Queue System (CBQS)**: a durable, off-chain messaging service for workloads running across Cowboy Runners. CBQS provides one ordered stream from which applications build work queues, publish/subscribe, chat rooms, request/reply, and replayable event logs. Broker-side **lanes** let many related rooms or documents share one stream without creating a chain object per room.

The Cowboy chain is a **governance and registry plane**. It records who owns a stream, which provider serves it, which key may authorize access to it, and whether the owner has paid. A provider-operated service, `cbqsd`, is the data plane: it stores records, manages consumer groups and delivery leases, pushes records to connected clients, and commits records durably in batches, signing one checkpoint per batch.

The two planes operate independently. Appending, delivering, acknowledging, and replaying a message require no chain transaction and no chain round trip. A broker serves from a periodically refreshed snapshot of the registry and keeps serving established sessions while the chain is unreachable. While the chain is reachable, chain authority reaches the data plane within one refresh interval; a compliant broker enforces grant expiry on every session whether or not the chain answers.

Payloads are opaque bytes to the broker. CBQS defines no key distribution; applications that need confidentiality encrypt before they append.

This paper explains the design and its rationale. CIP-39 and its native-room companion specify the wire formats, chain transitions, error codes, and conformance tests. This paper complements the **Cowboy Technical Whitepaper**, the **Cowboy Storage Whitepaper**, and the **Cowboy Secrets Whitepaper**; where it and the Technical Whitepaper conflict on shared platform behavior, the Technical Whitepaper prevails.

> **Deployment.** Cowboy Cloud runs on this protocol. Cowboy Chain, a public network on the same protocol, is under consideration for a future release.

Introduction

Cowboy actors communicate through consensus. That is the correct path when a message must change finality-bearing state, move value, or be visible to every validator. It is the wrong path for the high-volume coordination produced by persistent agents, collaborative applications, and off-chain services. Consensus messages are public, block-latency, gas-metered, and replicated by every validator even when only two workloads care about the payload.

© Cowboy Labs, Inc. 2026

Runners execute compute away from consensus. They also need shared messaging. Without one shared messaging service, a coordinator talking to several specialist agents builds persistence, retries, fan-out, cursors, leases, reconnection, backpressure, authorization, and crash recovery as application-specific infrastructure, and every chat room and collaborative document repeats the work.

CBFS solves durable object storage and CBSS solves authorized secret release; neither orders messages or tracks delivery. CBQS orders messages and tracks delivery at the broker. Cowboy records its tenancy, authorization, and billing.

CBQS is deliberately narrower than Redis, Kafka, or a general event-processing platform. It provides no key/value state, distributed locks, semantic routing, server-side payload search, platform-managed encryption, or exactly-once application effects. Applications use these services as follows:

- use actor messages and actor state for consensus effects;
- use CBQS for private or high-frequency off-chain coordination;
- use CBFS for large objects and for history beyond a stream's retention; and
- use CBSS, or any other key system, for the application keys that encrypt CBQS payloads.

Architectural Overview

Terminology

Term	Meaning
Stream	An ordered, append-only sequence of records with bounded retention, one assigned provider, and one chain-owned record.
Lane	A broker-side filtered view inside a stream, identified by a broker-assigned integer. Lanes share the stream's provider, billing, retention, and order.
Provider	An operator registered on chain to run `cbqsd`.
CBQS account	One prepaid balance per `(owner, provider)` pair, holding all billing state for the streams that provider serves for that owner.

© Cowboy Labs, Inc. 2026

Term	Meaning
StreamGrant	A credential signed by the stream's admin key that binds a holder key to verbs, lane and group scope, a time window, and rate caveats.
Consumer group	Broker-side delivery state in which consumers compete for each record of one lane under leases.
Registry snapshot	The broker's most recent read of finalized Stream Registry state, used for every authorization and billing decision until the next refresh.
Checkpoint	A provider-signed commitment to one contiguous range of durably committed records.

Key Features

1. Records, acknowledgements, leases, cursors, lanes, and groups stay between clients and `cbqsd`; the broker reads chain state only to refresh its snapshot.
2. One ordered stream supports queues, pub/sub, chat, request/reply, and replay. They share one stream, one sequence space, and one delivery model.
3. Every acknowledged append has crossed a synchronous durable commit; there is no provisional class.
4. Ownership, admin key, and an authorization generation live on chain and are applied from a snapshot; holder-bound grants authorize individual sessions; grant expiry is the hard bound on revocation.
5. One signed, hash-chained, gap-free checkpoint per committed batch gives a full-batch observer portable evidence of what the provider committed.
6. Subscriptions are credit-gated in encoded bytes; groups bound in-flight leases per group and per holder.
7. Each stream pays a flat rate per block from one prepaid account per `(owner, provider)` pair. Nothing is metered per message.

Differences vs. On-Chain Actor Messages

© Cowboy Labs, Inc. 2026

Aspect	Actor messages	CBQS
State authority	Consensus transition	Assigned provider, constrained by chain state
Payload visibility	Public to validators and history readers	Opaque to the broker; encrypted when the application encrypts
Write latency	Block inclusion and finality	Provider round trip plus one batch commit
Cost model	Per-transaction Cycles and Cells	Flat per-stream rent; no transaction per record
Ordering	Consensus order among actor inputs	One provider-assigned order per stream
Delivery	Actor invocation	Cursor replay or leased consumer group
Evidence	Consensus receipt	Provider-signed batch checkpoints
Correct use	Value, finality, public state	Private or high-frequency workload coordination

Related Work

Log Brokers: Apache Kafka

Kafka established the partitioned-log model: producers append, consumers retain positions, and consumer groups divide partitions among workers. Order holds within a partition rather than across a topic, which buys horizontal throughput at the cost of making applications choose partition keys.

CBQS chooses one total order per stream. Lanes filter that order. That gives up horizontal write scaling within one stream and makes agent coordination, chat history, and checkpoint verification simpler. The deeper difference is authority: Kafka authenticates against broker configuration, while CBQS checks a holder-bound grant against chain-recorded ownership, admin key, authorization generation, and solvency.

© Cowboy Labs, Inc. 2026

Durable Messaging: NATS JetStream, RabbitMQ, and Amazon SQS

JetStream provides stored streams, replay, push and pull consumers, acknowledgement policies, and retention. RabbitMQ quorum queues combine publisher confirms, consumer acknowledgements, dead-lettering, and replicated queues. SQS provides visibility timeouts and managed deduplication.

CBQS uses durable appends, leases, visibility timeouts, acknowledgements, and retry limits. It omits two common features. It has no dead-letter queue: a record whose attempts run out reaches a terminal state and stays in the log, replayable by sequence. It has no broker-side producer deduplication: a broker-side store is a per-append object that outlives the record it describes, and the consumer must make its effects idempotent under at-least-once delivery anyway.

Decentralized Messaging: Waku

Waku separates relay, filtering, historical store, and light-push protocols for decentralized messaging. Its store helps reconnecting clients recover missed messages but does not guarantee availability, and payload encryption is the application's.

CBQS trades decentralization for accountability. Each stream has one assigned provider, a funded account, and signed durable checkpoints. If that provider disappears, recovery depends on the application's own durable copy.

Tabular Summary

System	Ordering and delivery	Authority	Confidentiality and evidence
CBQS	One stream; cursor replay and leased groups	Cowboy chain + StreamGrant	Application encryption; signed batch checkpoints
Kafka	Partition; consumer groups and offsets	Broker deployment	Application policy; broker log
NATS JetStream	Stream or subject; stateful consumers	NATS account	Application policy; server acknowledgements

© Cowboy Labs, Inc. 2026

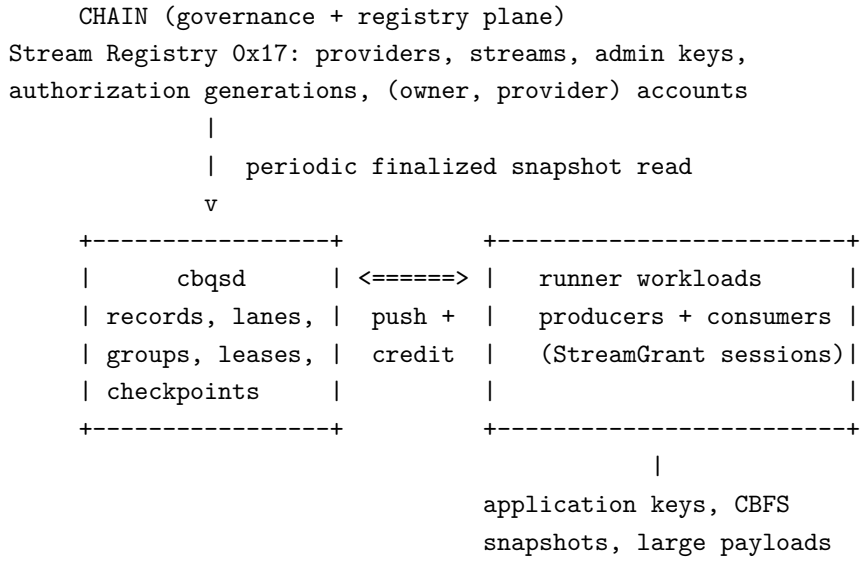
System	Ordering and delivery	Authority	Confidentiality and evidence
RabbitMQ quorum queue	Queue; acknowledgements and retries	Broker account and vhost	Application policy; publisher confirms
Amazon SQS	Queue or message group; visibility timeout	Cloud IAM	Service-managed; API responses
Waku	Content topic; relay, filter, and store	Peer and application policy	Application encryption; peer acknowledgements

Why a Dedicated Protocol

A provider may run an embedded log, a RocksDB-class store, or a mature broker underneath; `cbqsd`'s storage engine is pluggable. The externally visible system still has to verify grants against chain authority, enforce solvency and revocation from a snapshot, assign lane and group ids that are never reused, and produce a gap-free checkpoint chain. Those semantics are the client contract, and `cbqsd` is the layer that supplies them.

© Cowboy Labs, Inc. 2026

System Architecture



On-Chain Registry

The Stream Registry system actor at `0x17` holds three kinds of record and nothing else: provider records, stream records, and CBQS accounts. It never receives payloads, lane or group operations, leases, acknowledgements, or cursors. CBQS uses one top-level system opcode, `SYS_CBQS = 218`, carrying one versioned instruction family:

Instruction	Caller	Effect
`RegisterProvider`	Provider account	Create a provider record and pay the registration charge
`UpdateProvider`	Provider	Change endpoints, signing key, metadata, stream cap, or whether new streams are accepted
`CreateStream`	Owner	Settle the account, create a stream, and pay the creation charge
`UpdateStream`	Owner	Rotate the admin key and/or bump the authorization generation

Instruction	Caller	Effect
`CloseStream`	Owner, or provider when the account is insolvent	Settle, then close the stream
`TopUpAccount`	Any payer (only the owner may create the account)	Settle, then credit an `(owner, provider)` account
`SettleAccount`	Anyone	Settle elapsed rent; materialize suspension or revival
`WithdrawAccount`	Owner	Settle, then withdraw from an account with no live streams

Every handler reserves a fixed number of state reads and writes before its first check, every rejection leaves state unchanged and emits nothing, and each success emits exactly one event naming the provider, stream, or owner it touched.

Off-Chain Data Plane

`cbqsd` reads finalized registry state from a Cowboy node, decides every request from one snapshot, orders and stores records, runs consumer groups, enforces settings and rate limits, and signs checkpoints. It refreshes its snapshot every couple of seconds; when a refresh fails it keeps serving from the snapshot it has.

Clients connect directly to a provider endpoint over TLS WebSocket with binary frames. The Cowboy Gateway is not on the message path.

Trust Boundary

The chain is authoritative for stream existence, ownership, provider assignment, admin key, authorization generation, and the owner's balance and solvency. The provider is authoritative for the records, lanes, groups, and delivery state it has accepted. Clients are authoritative for payload meaning, encryption, idempotency, snapshots, and any projection back into consensus.

A provider can censor, delay, or destroy data. Checkpoints make some misbehavior detectable afterwards; they do not provide availability, and there is no on-chain dispute, staking, or slashing path. An application whose data must outlive its provider keeps its own durable copy.

© Cowboy Labs, Inc. 2026

Streams and Lanes

Stream Records

A stream record carries only what the chain must arbitrate:

Field	Type	Meaning
`chain_instance_id`	Bytes32	Genesis-configuration fingerprint of the chain
`stream_id`	Bytes32	Hash of the chain instance, owner, and owner nonce
`owner`, `owner_nonce`	Address, u64	Immutable; a nonce is never reusable, even after close
`admin_key`	Ed25519 key	Signs StreamGrants
`provider`	Address	Immutable assignment
`authorization_generation`	u64	Every grant names it; a bump invalidates all older grants
`status`	Active / Closed	`Closed` is absorbing

A stream record has no rate, balance, configuration, or suspension state. Economics live on the account and retention and rate settings live at the broker. Because the stream id commits to the chain instance, an id from one genesis history cannot address a stream in another that reuses the same numeric chain id.

Lanes Are Views, Not Partitions

Lane `0` exists in every stream, cannot be closed, and is where a client appends when it has no reason to route. Other lanes are created off chain by a holder with `LANE_ADMIN`; the broker assigns the next unused integer and never reissues one. A closed lane rejects new appends and stays replayable until retention passes its records.

Lanes preserve one global order. They have no independent provider, billing, retention, ordering authority, or quota: a hot lane consumes the stream's shared throughput and can backpressure its siblings. The broker enforces lane scope as an authorization boundary. Lane scope provides no cryptographic isolation. It does not hide traffic volume because sequences cover the whole stream. Tenants that must not observe each other use separate streams.

© Cowboy Labs, Inc. 2026

Two keyed forms of lane creation serve applications that need a stable mapping from their own identifiers to lanes. Keyed allocation binds an application key and metadata hash to one broker-assigned lane, so a retry returns the same lane. Room materialization derives a lane id deterministically from a room and seat, so every party computes the same lane without coordination.

Records

A record has a broker-assigned header containing the stream, lane, and stream-wide sequence starting at `1`. Its opaque payload can be up to the stream's message-size limit. The header carries no client-chosen field. Message ids, timestamps, reply routes, CBFS references, encryption framing, and application deduplication keys belong inside the payload. The broker never interprets payload bytes.

Delivery

Durable Append

The broker commits each stream's appends in batches: a batch closes after 20 ms with at least one record buffered, or at 1,024 records, and one synchronous durable write commits it. An append is acknowledged with its sequence only after its batch commits, never from memory or an operating system page cache. A batch is therefore the unit of both durability and signing.

The broker does not deduplicate appends. A producer whose append times out cannot learn whether it landed, and under at-least-once delivery the consistent choice is to retry and let consumers deduplicate on an identifier the producers put in the payload.

Subscriptions

A session opens subscriptions of two kinds:

- a **cursor** replays a lane scope from a sequence (inclusive) or from the head, in ascending order, and takes no leases; and
- a **group** subscription attaches to a consumer group and receives leased deliveries.

A start behind the retention floor returns `CursorTooOld` with the current floor, so the client can resume at an attested boundary.

© Cowboy Labs, Inc. 2026

Consumer Groups

A group is broker-side state over one lane. Its configuration names a start position, a visibility timeout and its ceiling, a group in-flight limit, a per-holder in-flight limit, an attempt limit, and a record deadline. The broker assigns group ids from a counter, as it does lane ids. A group orders its delivery cycles by sequence and may lease up to its in-flight limit at once; `max_in_flight = 1` gives serialized, single-consumer processing.

Each record reaches exactly one terminal state per group:

```
ACKED | EXPIRED
```

A lease names the record, attempt, holder, and expiry, and its handle is an unpredictable 32-byte value. Only the holder it was issued to may act on it:

Action	Effect
`Ack`	Terminal `ACKED`
`Nack`	Ends the attempt and frees the slot; the record is deliverable again when the lease would have expired
`Release`	Gives the lease back unprocessed and refunds the attempt, optionally after a short delay; at most eight refunds per record
`Extend`	Moves expiry forward, capped at the lease's issue time plus the maximum visibility
`Reject`	Terminal `EXPIRED`
(silence)	Expiry counts as an attempt exactly as `Nack` does

A record also expires when its attempts run out, when its record deadline passes, or when retention evicts it. There is no dead-letter queue and no redrive: an expired record stays in the log, where a cursor can replay it. `Nack` leaves the deadline unchanged, so answering is never worse for the group than silence. A holder uses `Release` for work it could not serve. It does not apply to work that failed.

At-least-once delivery repeats application effects. Consumers make their effects idempotent or deduplicate in their own state; neither an append acknowledgement nor an `Ack` makes an external side effect exactly once.

© Cowboy Labs, Inc. 2026

Flow Control

Each subscription carries a credit balance in encoded frame bytes, granted by the consumer and capped at 64 MiB. The broker never sends a delivery larger than the remaining credit; when credit is short of the next frame it says how many bytes are needed. A subscription that holds enough credit and accepts nothing for the slow-consumer timeout is closed, while one whose credit is merely too small is complying and is left alone. Scheduling across a connection's subscriptions is starvation-bounded, and connections cap open sessions and subscriptions.

Streams also carry two aggregate rate limits, append bytes and delivered bytes per second, enforced across every session and connection on the stream and answered with `RateLimited` rather than silent drops. A grant adds a per-holder append cap.

Authorization and Sessions

The owner controls the stream's admin key through a chain transaction. The admin key issues StreamGrants directly to workloads or users, off chain.

StreamGrant

A grant authorizes the holder key. Possession of the token alone confers no authority. It names the chain instance, stream, authorization generation, a nonce, the holder's Ed25519 key, a verb set, a lane scope (any, exact, or a sorted set), a group scope (any or exact), a validity window no longer than the maximum grant lifetime (24 hours by default), and message-size and append-rate caps.

Verb	Authority
`APPEND`	Append records within the lane scope
`CONSUME`	Open live subscriptions and group deliveries
`ACKNOWLEDGE`	Act on leases the holder was issued
`REPLAY`	Replay retained history
`GROUP_ADMIN`	Create, update, or delete groups — full control of a group
`LANE_ADMIN`	Create and close lanes
`STREAM_ADMIN`	Change the stream's broker settings

© Cowboy Labs, Inc. 2026

Scope is checked by intersection on every operation, lease actions included. Because lane and group ids come from counters, a narrow scope such as "lane 7" is a claim on whatever lane becomes the seventh; creating a lane or group therefore requires an unrestricted scope, so a narrowly scoped holder cannot mint the ids its own scope names.

Session Establishment

A session is a short handshake. The client sends its grant; the broker checks the chain instance and stream, the authorization generation, the admin signature, the time window, and the scope rules, cheapest checks first, and answers with a random challenge and session id. The client returns a proof signed by the holder key over the grant id, session id, and challenge. A captured grant is useless without the holder key, and a challenge completes at most one session.

Requests inside a session are not individually signed. Each carries the session id it executes under, and the broker binds it to that session's grant on the connection that opened it. Per-request signatures would cost a signature per message and would not stop an attacker positioned inside the TLS session from reordering or dropping frames.

Revocation and Chain Lag

The owner revokes by bumping the authorization generation, which every admin key rotation also does. The broker observes the bump at its next snapshot refresh and terminates affected sessions before serving their next request. Two bounds keep snapshot-based authority from becoming unbounded:

- a broker refuses new sessions when its snapshot is older than `cbqs.max_snapshot_age_blocks` (300 blocks by default); and
- every grant expires, and the broker terminates a session when its grant does.

Only the second holds against a broker that never refreshes, and a client can observe it directly. Grant expiry is therefore the guaranteed bound on a revoked holder, and short-lived grants are how an owner narrows it. There is no per-grant revocation: a generation bump invalidates every outstanding grant on the stream.

Native Room Sessions

Rooms executed natively by runners use a second way to open a session. Instead of an admin-signed grant, the runner assigned to a room seat signs a delegation for the seat's workload key, accompanied by a finalized-state proof that the room, seat, job, and native session are authorized.

© Cowboy Labs, Inc. 2026

The broker derives the permitted verbs and the exact room lanes from those records; scopes supplied by the requester carry no authority. Each room has a shared messages lane and each seat an inbox lane, both at deterministic ids.

The same extension defines a content-free accounting read. A runner proves its original seat authority and the session's current billing eligibility and receives the group's durable terminal prefix for its inbox. The prefix shows how far delivery has settled. The read returns no messages and opens no session. Runners use it to report earned usage after an interrupted acknowledgement without repeating work or billing for work they did not do.

Durability and Checkpoints

Each committed batch produces one checkpoint, signed by the provider:

Field	Meaning
`chain_instance_id`, `stream_id`, `provider`	What the checkpoint is about and who signed it
`first_sequence`, `last_sequence`	The contiguous range committed
`records_digest`	Hash over the ordered record digests and the record count
`previous_checkpoint`	Id of the predecessor, or a stream-specific genesis value
`committed_at_ms`	Commit time, non-decreasing along the chain
`snapshot_height`	A finalized height the provider read; selects the signing key

Checkpoints form a hash chain that covers the stream without gaps: each range starts one past its predecessor's end, the first starts at sequence `1`, and each names its predecessor's id. Linkage and coverage are one check, so a provider cannot link two ranges and silently drop the records between them, or swap a receipt for another over the same range.

Clients request the latest checkpoint or one identified by id. The broker does not push checkpoints. A subscriber holding every record of a batch recomputes its digest; a mismatch, a broken link, a gap, or two checkpoints over one range is signed evidence of provider misbehavior.

The digest covers the whole batch, so only a party holding every record in it can verify; a lane-scoped subscriber or a competing group consumer cannot. No consensus state anchors a

checkpoint chain, so a provider serving two divergent chains is caught only by a party holding both, and clients that care retain their latest checkpoint id outside the provider. Checkpoints are retained only as long as the records they attest, and they count against the stream's byte limit.

A provider record keeps its current signing key and one previous key, each with the height from which it was valid. A verifier picks the key by the checkpoint's `snapshot_height`, which gives three verdicts: valid, invalid when a resolved key rejects the signature, and unverifiable when the checkpoint predates the rotation before last. One rotation therefore never costs a provider its history, and a client that wants a checkpoint checkable across many rotations retains the key alongside it.

Billing

CBQS charges a flat rate for the existence of a stream. It meters no messages, bytes, or capacity: the provider's exposure is bounded instead by the per-stream settings ceilings the broker enforces and by the provider's own stream cap.

Accounts

All economic state lives on one account per `(owner, provider)` pair:

Field	Meaning
`balance`	Nano-CBY held in registry custody at `0x17`
`active_streams`	The account's open streams
`rate_per_block`	Per-stream rate, copied from governance when a run of streams begins
`last_settled_block`	Height rent has been settled through
`suspended`	Set when settlement found the balance short

The broker tests solvency on the account that pays it, using the same expression and fields as the chain. Settlement has one recipient, so its cost is fixed. A provider's collectible is isolated from the same owner's streams at other providers.

Only the owner can create the account. Its first top-up must be at least 1 CBY. It then creates streams against that account. The account is deleted when its balance and stream count both reach zero.

© Cowboy Labs, Inc. 2026

Settlement

Every instruction that touches an account settles it first, then applies its own effect, then finalizes:

```
due = (H - last_settled_block) * rate_per_block * active_streams
```

Settlement pays `due` if the balance covers it. Otherwise it pays the whole balance, writes off the remainder, and suspends the account. A suspended account accrues nothing, so arrears cannot grow and revival never costs more for having waited. A top-up settles against the old balance before crediting new funds. New money cannot pay for a written-off interval, and one top-up revives a funded account. Settlement is permissionless and a pure function of the account and height, so whoever submits it chooses only when the same arithmetic runs.

Each paid amount is split by a protocol constant: 90% to the provider and the remainder to the Platform Fee Account at `0x18`. The split is a constant rather than a governance parameter because settlement pays for elapsed time, and a live parameter would let one governance write reprice service already rendered. For the same reason the rate is copied into the account when its stream count goes from zero to one, a moment only the owner controls, and stays fixed for that run of streams.

Solvency at the Broker

The broker refuses, with `AccountSuspended`, every request that creates or extends durable state, appends, lane and group creation, settings changes, and lease extension, when the account cannot cover rent through its snapshot height. It issues no new leases while the account is unserviceable, but `Ack`, `Nack`, `Release`, and `Reject` stay available so consumers can drain what they hold. Reads and replays are the provider's choice. The test runs on every request, not only at session open, so a session opened while solvent cannot keep appending for the life of its grant. The test reads the broker's current snapshot, so a suspension recorded on chain takes effect at the next successful refresh.

Charges and Exit

Two one-time charges price permanent registry state: creating a stream and registering a provider each cost 5 CBY, debited from the caller's native account and burned to the zero address, so a self-providing owner cannot recover them through the rent split. Both are protocol constants that no governance write can zero.

A provider leaves by refusing new streams and letting its streams close. When an owner stops paying, the provider may close that owner's streams once the account, after settlement, cannot

© Cowboy Labs, Inc. 2026

fund another block, which releases the provider's capacity without seizing anything the owner did not owe. A provider cannot terminate service to an owner who keeps paying. If the Stream Registry is paused, rent keeps accruing, so a pause consumes at most each account's balance; recovery is one top-up per account.

Provider assignment is immutable. Moving an application to another provider means creating a new stream there, copying or replaying data from a durable source the application controls, issuing new grants, and closing the old stream only after the new provider's latest checkpoint covers everything the old one held.

Retention and Stream Settings

Each stream has five broker-side settings, changed by a `STREAM_ADMIN` holder without a chain transaction and validated against governance ceilings when set: retention time, retained bytes, maximum message size, append rate, and delivered rate. A later governance change constrains the next settings change. It leaves existing settings in force.

The retention floor advances past a record when either bound is reached: the retention time has elapsed since the broker committed it, or keeping it would exceed the byte limit. Records and their checkpoints count against the limit. The clock is the broker's commit time, never a client timestamp. Eviction is a terminal transition: every non-terminal delivery cycle for an evicted record becomes `EXPIRED`, and a group's record deadline cannot exceed the stream's retention. The floor is monotone.

Lowering a setting is destructive. A lower retention time or byte limit advances the floor at once and can discard nearly the whole retained log in one request, and revoking the grant afterwards restores nothing. The mitigation is that `STREAM_ADMIN` is a separate verb an owner need not issue.

A closed or suspended stream retains data at the provider's discretion, stated in its registered metadata. Applications that need history beyond retention, or recovery independent of the provider, write snapshots to owner-controlled CBFS volumes; CBQS neither creates nor trusts those snapshots. Large payloads follow the same split: the application stores the object in CBFS and appends a reference.

Application Patterns

© Cowboy Labs, Inc. 2026

Pattern	Construction
Work queue	One lane + one consumer group; consumers compete under leases
Fan-out	One group per independent consumer, or cursor replay
Serialized processing	A group with `max_in_flight = 1`
Chat room	One lane per room; cursor replay for history; attachments in CBFS
Agent room	Native room sessions: shared messages lane plus a group-consumed inbox lane per seat
Request/reply	Correlation and reply lane inside the payload
Collaborative document	One stream per workspace, one lane per page; encrypted updates; snapshots in CBFS
Event log	Cursor replay within retention; CBFS snapshots beyond it

CBQS never invokes a consensus actor. A client may consume a stream, derive a projection, and submit an ordinary actor transaction; that projection and its timing then become public consensus data.

© Cowboy Labs, Inc. 2026

Security Considerations

Confidentiality and Metadata

Payloads are opaque bytes to the broker, and the broker receives no key. Confidentiality is exactly as strong as the application's own encryption and key management. The provider sees which workloads connect; stream, lane, and group identifiers; message sizes, timing, and volume; and delivery state. The chain exposes stream existence, owner, provider, billing activity, and authorization-generation changes. Because sequences are stream-wide, any holder on any lane can measure how many records the whole stream has taken.

Sessions and Revocation

An attacker who can inject into an established TLS session acts with that session's authority until it closes. Holder keys should be workload-scoped and grants short-lived. Chain authority reaches an established session at the next successful refresh and not at all during a chain outage; grant expiry is the only bound that does not depend on the broker, and owners rotating away from a compromised holder should treat it as the guarantee.

Provider Identity

The handshake authenticates the client to the broker. It does not authenticate the broker to the client. The client relies on TLS to reach the host its registered endpoint names. An attacker using a mis-issued certificate or a DNS or BGP hijack can intercept the connection. It lacks the provider's signing key, so any full-batch observer can detect its invalid checkpoints. A compromise of the provider's chain account is worse: one update can repoint the endpoint and install a new key. The previous key survives one such rotation, so a client that pinned a height beforehand can detect it until a second rotation. A relay of a whole handshake to the genuine broker is not addressed, because the proof carries no channel binding.

Clients apply a dialing policy to every destination they connect to: no loopback, link-local, private, or special-use addresses, every address a name resolves to is checked, the client connects only to an address it checked, and redirects are never followed. Consensus admits endpoint URIs on byte rules alone and does not claim they are safe to dial.

Destructive Verbs and Co-Tenancy

`STREAM_ADMIN` can discard the retained log and `LANE_ADMIN` can permanently close lanes; `GROUP_ADMIN` is full control of a group. A holder with `ACKNOWLEDGE` can reject records it was

leased, and a silent consumer walks the records it holds to `EXPIRED` in `max_attempts` visibility timeouts; `max_in_flight_per_holder` bounds how many at a time. A producer at its permitted rate can fill the byte limit and evict every consumer's backlog. Egress is bounded per stream, not per holder. Each of these is a trust decision the owner makes when it issues a grant; applications that cannot make it give the untrusted party its own stream.

Provider Failure

The assigned provider can censor, delay, or destroy data, and there is no staking, slashing, dispute, or reassignment. Checkpoints turn conflicting signed histories into evidence but do not make lost bytes recoverable. The remedy is the application's own durable copy and a move to a new stream.

Billing Exposure

Solvency is tested per request against the broker's current snapshot. While refreshes succeed, a provider's exposure to an unpaid owner is bounded by the service it renders within one refresh interval. During a chain outage the snapshot stops advancing and established sessions keep being served from it: `cbqs.max_snapshot_age_blocks` refuses only new sessions, so it does not bound this exposure. What remains is grant expiry, which ends every session, and the stream's rate and retention limits, which bound what a session can do before it ends. Settlement cadence does not change what a provider collects. Flat pricing cross-subsidizes heavy streams with light ones; a provider unwilling to serve at the governance rate stops accepting new streams.

Denial of Service

Frames are capped at 5 MiB and every variable-length field is capped at decode, before allocation and before signature verification. Unauthenticated peers are bounded by a per-challenge deadline and by closing any connection that completes no session, or holds a partial message, past the handshake timeout. Lane and group creation are rate-limited per stream, lanes, groups, sessions, and subscriptions are capped, and credit saturates. Permanent registry state costs unrecoverable capital: the stream creation and provider registration charges, plus a top-up floor on account creation.

Chain Resets

Every signed object and every stream id commits to the chain-instance id, so an object from one genesis history is not accepted by another that reused the same numeric chain id. Decoders

© Cowboy Labs, Inc. 2026

reject unknown versions, tags, and algorithms, and a broker refuses to open a store whose format marker it does not recognize.

Parameters

Selected defaults. Governance changes are Tier 0; ceilings apply when a setting or group is configured and never retroactively.

Parameter	Default	Meaning
`cbqs.stream_rate_per_block`	80,000 nano-CBY	Flat rent per stream per block
`cbqs.max_streams_per_account`	1,024	Open streams per `(owner, provider)` account
`cbqs.snapshot_refresh_ms`	2,000	Snapshot refresh interval
`cbqs.max_snapshot_age_blocks`	300	Oldest snapshot that may open a session
`cbqs.max_clock_skew_ms`	60,000	Accepted grant clock skew
`cbqs.retention_ms`	604,800,000	Retention ceiling (7 days)
`cbqs.max_retained_bytes_per_stream`	1 GiB	Retained-bytes ceiling
`cbqs.max_message_bytes`	256 KiB	Message-size ceiling
`cbqs.max_append_bytes_per_sec`	1 MiB/s	Append-rate ceiling
`cbqs.max_delivered_bytes_per_sec`	4 MiB/s	Delivered-rate ceiling
`cbqs.visibility_ms`	30,000	Maximum lease visibility
`cbqs.max_attempts`	10	Maximum attempts per record per group
`cbqs.max_in_flight_per_group`	4,096	Maximum group in-flight limit
`cbqs.max_groups_per_stream`	4,096	Active groups per stream
`cbqs.max_lanes_per_stream`	16,384	Active lanes per stream
`cbqs.max_lane_creates_per_min`	256	Lane creation rate per stream

© Cowboy Labs, Inc. 2026

Parameter	Default	Meaning
`cbqs.max_group_creates_per_min`	64	Group creation rate per stream
`cbqs.commit_batch_max_ms`	20	Batch close time
`cbqs.commit_batch_max_records`	1,024	Batch close size
`cbqs.max_wire_frame_bytes`	5 MiB	Frame ceiling (fixed)

Protocol constant	Value
Stream creation charge	5 CBY, burned
Provider registration charge	5 CBY, burned
Minimum account-creating top-up	1 CBY, refundable
Provider share of rent	90%
Refunded releases per record	8

Relationship to the Cowboy Whitepapers

This document layers on, and never supersedes, the Cowboy Technical Whitepaper:

- Actors, actor messages, execution, finality, and public state remain governed by the Technical Whitepaper. CBQS never turns a record into a consensus effect without an ordinary actor transaction.
- Cycles, Cells, balances, transaction authorization, system actors, and governance follow the Technical Whitepaper; CBQS instructions consume them normally.
- The Storage Whitepaper governs CBFS volumes used for large payloads, snapshots, and history beyond retention. A CBFS reference inside a payload does not change CBFS authorization or durability.
- The Secrets Whitepaper governs CBSS. Applications may use it to manage the keys that encrypt their payloads; CBQS itself neither distributes keys nor depends on CBSS.
- Runner assignment and workload execution follow the Technical Whitepaper. A StreamGrant authorizes queue operations, not execution or spending.

CIP-39 is the specification for canonical encodings, signing preimages, error codes, metering, and conformance. The shared protocol codec and released node and broker are its executable form.

© Cowboy Labs, Inc. 2026