

Cowboy Queues: CBQS

Technical Whitepaper

Charles DePue, Cowboy Foundation

August 6, 2026 — Draft v0.1.0

Abstract

This document specifies the **Cowboy Queue System (CBQS)**: a protocol for chain-anchored, off-chain messaging among workloads running across Cowboy Runners. CBQS provides one ordered stream primitive from which applications construct work queues, publish/subscribe, chat rooms, request/reply, replayable event logs, and collaborative document transports. Broker-side **lanes** let related rooms or documents share one stream, provider, retention policy, billing position, and encryption domain without creating one chain object per room.

The Cowboy chain is the control plane. It records stream ownership, provider assignment, configuration, pricing, escrow, authorization generation, and encryption generation. A conforming provider-operated **cbqsd** service is the data plane. It accepts and stores records, manages consumer groups and delivery leases, pushes records to connected clients, and signs receipts. Append, delivery, acknowledgement, replay, and cursor movement require no chain transaction.

CBQS defines two delivery classes. **Standard** acknowledges an append only after the record, idempotency result, and receipt-chain state cross the provider's durable commit boundary; it supports at-least-once consumer groups, leases, retries, dead-lettering, and redrive. **Fast** may acknowledge before durable commit; it exposes provisional and durable watermarks, signed checkpoints, and explicit clean-or-void restart transitions so repairable applications can recover without confusing low latency with durability.

CBQS supports streams configured for client-side end-to-end encryption. For those streams, conforming clients encrypt application records; a stream's admin issues holder-bound **Stream-Grants**; clients prove possession of the holder key and sign each request; and the design uses CBSS to escrow a stream root for an authorized rotation workload. Members receive Hybrid Public Key Encryption (HPKE)-wrapped generation keys through a linked key stream. The provider still sees stream, lane, group, size, timing, traffic, and connection metadata. The wire protocol does not prove that submitted payload bytes are encrypted. Signed receipt chains make conflicting signed histories and failure to serve acknowledged data evident to a receipt-holding client. They do not prove continued retention, distinguish loss from withholding, or force availability.

This specification complements the **Cowboy Technical Whitepaper**, the **Cowboy Storage Whitepaper**, and the **Cowboy Secrets Whitepaper**. Where this document and the Technical Whitepaper conflict on shared platform behavior, the Technical Whitepaper prevails. The numbered Core Protocol below consolidates the currently agreed CBQS behavior. A conforming release requires this paper, the shared protocol codec, and the released node implementation to agree.

Draft status. CBQS is a protocol design, not a currently available Cowboy network service. Present-tense normative statements define conformance; they do not claim a hosted endpoint, deployed network, client library, application integration, or measured service level. The recipient and split for accrued rent remain an open design decision in this draft.

Introduction

Cowboy actors communicate through consensus. That is the correct path when a message must trigger finality-bearing state, move value, or be visible to every validator. It is the wrong path for the high-volume coordination generated 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 need the payload.

Runners move compute away from consensus, but moving compute does not create a shared communications substrate. A coordinator talking to several specialist agents otherwise has to build persistence, retries, fan-out, cursors, leases, reconnection, backpressure, authorization, encryption, and crash recovery as application-specific infrastructure. Chat rooms and collaborative documents repeat much of the same work.

CBFS solves durable object storage. CBSS solves authorized secret release. Neither supplies message ordering or delivery state. CBQS is the corresponding messaging layer: a bounded hot log whose ownership and economics are anchored to Cowboy while its payload path remains off chain.

CBQS is deliberately narrower than Redis, Kafka, or a general event-processing platform. It does not provide arbitrary key/value state, distributed locks, semantic routing, server-side payload search, or exactly-once application effects. It provides the minimum shared machinery needed for Cowboy workloads to exchange private, replayable, flow-controlled messages under chain-derived authority.

The result is a clean boundary:

- 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 history beyond the broker's hot window; and
- use CBSS to authorize recovery of the root that drives membership rotation.

Architectural Overview

The material from **Architectural Overview** through **Application Patterns** is descriptive and non-binding. **Core Protocol** and §§1–13 use the key words **MUST**, **MUST NOT**, **REQUIRED**, **SHOULD**, **SHOULD NOT**, and **MAY** as described by RFC 2119 and RFC 8174 when, and only when, they appear in all capitals.

Terminology

- **Stream:** One ordered, append-only sequence of records with bounded retention, one assigned provider, and one chain-owned lifecycle.
- **Lane:** An opaque broker-side filtered view inside a stream. Lanes share the parent stream's order, provider, quota, rent, retention, authorization generation, and encryption generation.
- **Main stream:** The application-data stream selected by a client.
- **Key stream:** The linked **standard** stream that stores signed membership batches and HPKE-wrapped data-key envelopes for an encrypted main stream.
- **Provider:** An operator registered on chain to run `cbqsd` for assigned streams.
- **StreamGrant:** A directly admin-signed credential binding one holder key to verbs, lane and group scope, time bounds, and resource caveats.
- **Consumer group:** Durable broker-side state that delivers each applicable **standard** record to one group member under a lease.
- **Standard delivery:** Durable-before-ack append plus receipt-backed replay and consumer-group delivery.
- **Fast delivery:** Provisional low-latency append plus durable checkpoints and explicit restart lineage.
- **Finalized view:** One immutable chain snapshot used for every authorization and economic decision made while processing a broker request.
- **Receipt:** A provider-signed commitment to an append, delivery-state mutation, checkpoint, key batch, or retention boundary.

Key Features

CBQS combines eight features that are normally split across application code, cloud queue products, and chain contracts:

1. **No consensus write on the message path.** Records, acknowledgements, leases, cursors, and group operations remain between clients and `cbqsd`.

2. **One ordered primitive.** Queues, pub/sub, chat, request/reply, replay, and document update logs use the same stream and receipt model.
3. **Two explicit durability classes.** Standard waits for durable commit; Fast exposes a bounded provisional suffix and never labels it durable.
4. **Chain-derived authority.** Stream lifecycle and generations come from finalized chain state, while holder-bound grants authorize individual data operations.
5. **Client-side content encryption.** On encrypted streams, conforming clients keep generation data keys out of the broker data plane. Membership rotation is rooted in CBSS and delivered through HPKE envelopes; the broker does not enforce that arbitrary submitted bytes are ciphertext.
6. **Signed provider evidence.** Global, per-lane, per-group, checkpoint, and retention receipt chains let clients retain portable anchors.
7. **Explicit flow control.** Every subscription begins with zero record and byte credit; quotas, leases, and starvation-bounded scheduling limit fan-out pressure.
8. **Native economics.** The chain escrows fixed reserved-capacity rent and supports either a provider quote or a governance base schedule without trusting a per-message usage report.

Differences vs. On-Chain Actor Messages

Aspect	Actor messages	CBQS
State authority	Consensus transition	Assigned off-chain provider, constrained by chain state
Payload visibility	Public to validators and history readers	Client-side encryption on encrypted streams; metadata remains visible
Write latency	Block inclusion and finality	Provider round trip; class-dependent durable boundary
Cost model	Per-transaction Cycles and Cells	Fixed stream rent; no transaction per record
Ordering	Consensus order among actor inputs	One provider-assigned stream order
Delivery	Actor invocation	Replay, push, cursor, or leased consumer group

Aspect	Actor messages	CBQS
Failure evidence	Consensus receipt	Provider-signed receipt and checkpoint chains
Correct use	Value, finality, public state	Private or high-frequency workload coordination

Related Work

Log Brokers: Apache Kafka

Apache Kafka established the durable partitioned-log model: producers append events, consumers retain positions, and consumer groups divide partitions among workers. Kafka guarantees order within a partition rather than across an entire multi-partition topic. That is the right trade for horizontal throughput, but it pushes applications to pick partition keys and reason about cross-partition order.

CBQS v1 chooses one total order per stream and treats lanes as filtered views, not partitions. This costs horizontal write scaling within one stream but makes agent coordination, chat history, and receipt verification simpler. The more fundamental difference is authority: Kafka authenticates against broker configuration, while CBQS evaluates a holder-bound grant against finalized Cowboy ownership, generation, provider, status, and escrow state.

Durable Messaging: NATS JetStream, RabbitMQ, and Amazon SQS

NATS JetStream provides stored streams, replay, push and pull consumers, acknowledgement policies, retention, and at-least-once delivery. RabbitMQ quorum queues combine publisher confirms, consumer acknowledgements, dead-lettering, and replicated durable queues. Amazon SQS FIFO queues provide managed ordered message groups and deduplication.

CBQS borrows the useful delivery vocabulary—durable append, leases, acknowledgements, retry limits, dead letters, and redrive—but not the service account as the root of trust. A CBQS provider must verify finalized chain authority, enforce StreamGrant caveats, issue canonical receipt chains, and surface encryption and authorization generation changes. Clients speak the CBQS protocol, not the provider’s internal storage-engine protocol.

Decentralized Messaging: Waku

Waku separates relay, selective filtering, historical store, and light-push protocols for decentralized messaging. Its Store protocol helps reconnecting clients recover missed messages but does not itself guarantee data availability, and payload encryption is left to the application.

CBQS makes a different trade. Each stream has one explicitly assigned provider, a prepaid capacity position, an application-defined encryption domain, and signed durable anchors. That improves accountability and Cowboy integration but does not provide decentralized availability: if the assigned provider disappears, recovery depends on a verified application snapshot, another stream, or client-held state.

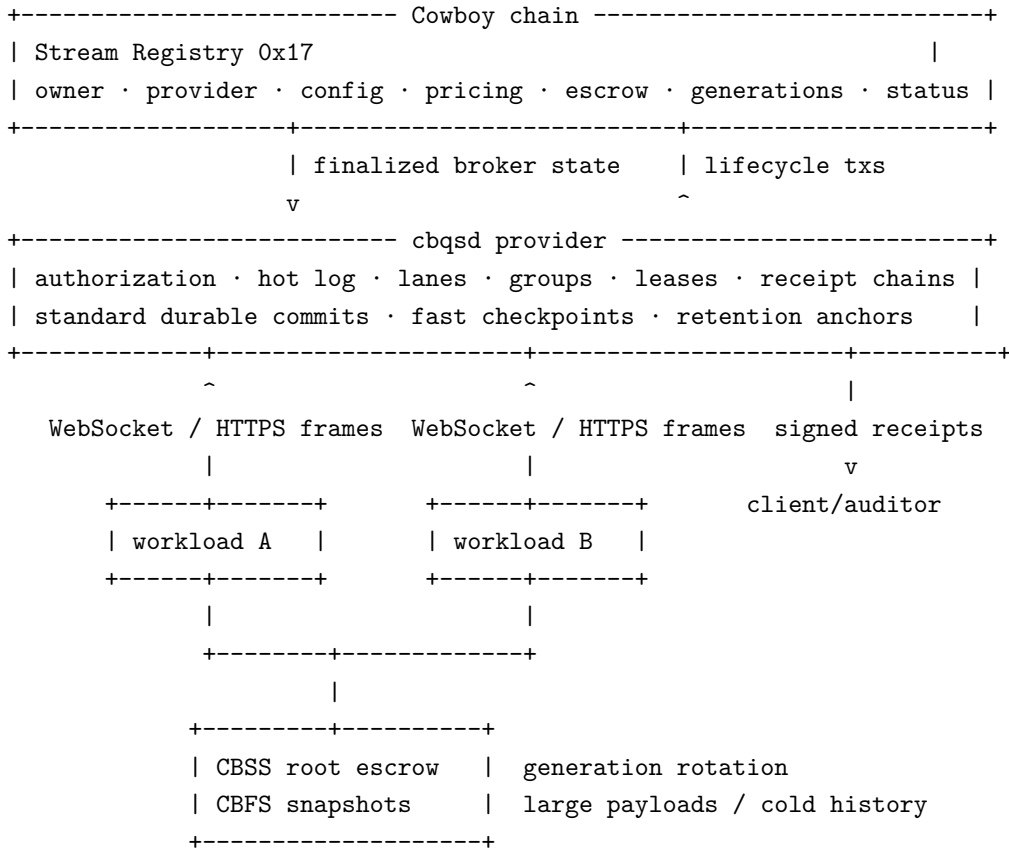
Tabular Summary

System	Ordering and delivery	Authority	Confidentiality and evidence
CBQS	One stream; Standard groups and Fast replay	Cowboy chain + StreamGrant	Client-side encryption profile; signed receipt chains
Kafka	Partition; consumer groups and offsets	Broker deployment	Application payload policy; broker log and offsets
NATS JetStream	Stream or subject; stateful consumers	NATS account	Application payload policy; server acknowledgements
RabbitMQ quorum queue	Queue; acknowledgements and retries	Broker account and vhost	Application payload policy; publisher confirms
Amazon SQS FIFO	Message group; visibility timeout and delete	Cloud IAM	Service-managed protection; API responses
Waku	Content topic; relay, filter, and store	Peer and application policy	Application encryption; peer acknowledgements

Why a Dedicated CBQS Protocol

A provider may use an embedded segmented log, RocksDB-class store, or a managed queue engine internally. The externally visible system still has to verify StreamGrants, bind every request to one finalized view, enforce generation revocation and stream economics, preserve canonical receipt chains, and implement Fast lineage transitions. `cbqsd` is the protocol layer that supplies those semantics. The internal engine is not the client contract.

System Architecture



On-Chain Control Plane

The Stream Registry system actor at 0x17 owns provider records, stream records, configuration bounds, pricing basis, rent escrow, and lifecycle events. It does not receive record payloads, consumer groups, delivery leases, acknowledgements, or cursors.

A stream record snapshots the assigned provider and its receipt-signing key, the delivery class, the encryption choice, the pricing basis and rate, and the initial bounded configuration. Mutable fields include selected capacity and retention settings, admin and rotation keys, escrow, and the authorization and encryption generations. Provider assignment, delivery class, main/key purpose, and encryption choice are immutable for the life of a stream.

Off-Chain Data Plane

cbqsd reads an atomic finalized broker-state view, verifies grants and request proofs, orders records, enforces quotas, persists delivery state, and signs receipts. It binds its durable store to

one chain instance and refuses to serve when the connected node reports a different genesis-derived chain identity.

Clients connect directly to the provider. The Cowboy Gateway is not on the message path. Registered provider endpoints use TLS WebSocket and HTTPS long-poll; both bindings carry the same canonical objects and semantics. TLS termination is an operator deployment boundary rather than a change to the CBQS protocol.

Trust Boundary

The chain is authoritative for ownership, provider assignment, configuration, pricing, escrow, status, and generations. The provider is authoritative for the records and consumer state it accepted, subject to its signed receipts. Clients are authoritative for payload meaning, decryption, application-side idempotency, snapshots, and any projection back into consensus.

The provider can censor, delay, withhold, or destroy records. Receipts make some conflicting histories and loss detectable; they do not replicate the data or force the provider to answer. Provider staking, slashing, transparent replication, and in-place reassignment are not properties of CBQS v1.

Streams, Lanes, and Ordering

Main Streams and Key Streams

An unencrypted stream is one main stream. An encrypted stream is created atomically with a linked **standard** key stream under the same owner and provider. The key stream carries signed rotation batches and HPKE ciphertexts, not plaintext keys. Its lifecycle mirrors the main stream, and the main stream's escrow and pricing cover the pair.

Each main stream has one global append order. Standard record identifiers are (**provider_epoch**, **sequence**); Fast identifiers add **broker_epoch** so records from a provisional lineage cannot be confused across restarts. Every v1 **provider_epoch** is zero. A nonzero value is invalid rather than an in-place provider migration signal.

Lanes Are Views, Not Partitions

The all-zero lane is present in every main stream. An authorized holder may create additional lane ids off chain. Lane creation and closure append provider-originated events with signed

append receipts to the default lane so stream-wide clients can discover topology changes without polling; bounded lane enumeration provides recovery after those events fall behind retention.

Lanes preserve one global order and also maintain a per-lane receipt predecessor. An exact-lane consumer can therefore verify continuity without downloading unrelated payloads. Lanes do not have independent rent, retention, providers, ordering authorities, or quotas. A hot lane can backpressure its siblings.

Lane scope is an authorization boundary at the broker, not a cryptographic boundary. Every lane in an encrypted parent uses the same generation data key. Applications use a separate stream or independent application-layer encryption when two lanes require different confidentiality domains.

Delivery Classes

Property	Standard	Fast
Append acknowledgement	After atomic durable commit	May be provisional before batch commit
Record receipt	Append receipt per record	Application records enter later checkpoints; durable provider events receive append receipts
Consumer groups	Yes	No
Leases / ACK / NACK / redrive	Yes	No
Crash behavior	Successful append survives restart under honest storage	Provisional suffix may be voided after crash
Correct workload	Jobs, chat history, commands, durable events	Conflict-free replicated data type (CRDT) operations, presence, typing, telemetry, cursors

Standard Delivery

A successful Standard append means that record bytes, the idempotency entry, the exact retry response, the global and lane receipt predecessors, and the new receipt tails crossed one atomic

durable commit boundary. The provider does not acknowledge from process memory or an operating-system page cache.

Idempotent Append

The deduplication key is the tuple (`stream_id`, `holder_signing_key_id`, `client_message_id`). Within the configured horizon, an identical retry returns the original receipt. Reusing the key with different canonical request bytes, including a different lane, is an `IdempotencyConflict`. After the horizon, the broker reports `IdempotencyExpired` instead of silently appending again.

Consumer Groups

Groups are durable broker objects, not chain state. A group selects one exact lane and either strict FIFO or concurrent delivery. Each applicable record reaches one terminal state per delivery cycle:

ACKED | DEAD_LETTERED | EXPIRED

Delivery creates a lease with an attempt number and expiry. ACK completes the cycle; NACK schedules a bounded retry; EXTEND moves the lease expiry within a maximum visibility window; REJECT dead-letters immediately; exhaustion of the attempt limit dead-letters automatically. Redrive creates a new cycle at the group tail and retains evidence of the earlier terminal state.

Strict FIFO exposes only the lowest nonterminal cycle. Concurrent mode may lease later records up to the group's in-flight limit. A second limit applies per holder across every group, subscription, and connection on the stream, so a holder cannot multiply its `StreamGrant` allowance by opening more sessions.

Fast Delivery

Fast is for application logs that can repair a bounded crash-loss window. A provider may return a provisional acknowledgement before durable commit, but stops extending the provisional suffix and commits when either the stream's batch-time or batch-record threshold is reached.

Clients observe two watermarks:

- the **provisional watermark**, which is the highest record emitted in the current broker epoch; and
- the **durable watermark**, which is the highest record covered by a signed checkpoint.

A client does not persist a provisional position as durable. Snapshots and cross-device recovery anchors cover only checkpointed records.

Broker Epochs and Repair

Before accepting records after startup, the broker durably advances the stream's broker epoch and publishes one signed transition. A **Clean** transition attests that the prior provisional and durable watermarks matched at shutdown. A **Void** transition invalidates every prior-epoch record above the last durable checkpoint. Conflicting transitions are provider-equivocation evidence.

Applications using Fast need stable operation ids or an idempotent merge model. A Yjs client, for example, restores state through the last durable checkpoint, discards a void suffix, computes a state-vector diff from local state, and re-appends that diff. A conforming client surfaces the transition; it never silently reconnects past an unexplained epoch change.

Transport and Flow Control

The canonical persistent binding is a bidirectional TLS WebSocket with binary frames. HTTPS long-poll is a required fallback and preserves the same grants, proofs, cursors, leases, receipts, and response bounds. Consumers connect outbound; CBQS requires no inbound workload route.

Every subscription starts with zero credit. The consumer grants record and byte credit, and a provider may send only when both counters cover the next record. Standard leases begin when the delivery frame is emitted. Providers also cap subscriptions per connection and schedule active subscriptions with deficit round robin or an equivalent starvation-bounded policy. Closing a slow subscription does not acknowledge its records; any leases expire under the ordinary group rules.

Authorization and Sessions

CBQS separates chain ownership from data-plane use. The owner controls the stream's admin key through a chain transaction. The current admin key then issues StreamGrants directly to workloads or users without putting each grant on chain.

StreamGrant

A grant binds authority to a holder rather than making bearer possession of a token sufficient:

```
StreamGrantV1 {  
    context  
    stream_id  
    authorization_generation
```

```

grant_nonce
holder_signing_key
holder_hpke_key?
verbs
lane_scope
group_scope
not_before_ms, expires_at_ms
max_message_bytes
max_append_bytes_per_sec
max_in_flight
max_lane_creates_per_min
admin_signature
}

```

The seven v1 verbs are:

Verb	Authority
APPEND	Append an application record within scope
CONSUME	Open live delivery within scope
ACKNOWLEDGE	Mutate a matching delivery lease
REPLAY	Read retained history within scope
GROUP_ADMIN	Create, update, or delete a scoped group
REDRIVE	Create a successor cycle for a terminal record
LANE_ADMIN	Create or close lanes within the grant caveats

The grant is valid only in the named chain instance and stream, at the current authorization generation, within its time window, and within the intersection of stream, provider, protocol, and grant limits. Unknown verb bits are invalid.

Proof of Possession

Opening a session is a two-step handshake. The provider returns a random challenge and unique session id. The holder signs the challenge together with the grant id, chain context, stream, provider epoch, and session id. A captured grant is therefore insufficient without the holder's private key.

Every later request carries a proof over the session id, a strictly increasing counter, a canonical method tag, and the hash of the canonical request body. The method in the proof must match

the method encoded in the body. Transport correlation ids, WebSocket framing, and HTTPS framing do not enter the signed body hash.

The session counter prevents replay within a connection. The body hash prevents a valid signature for one operation from being moved to another. Strict canonical encoding prevents two components from signing different byte interpretations of the same apparent object.

Finalized-View Authorization

For each operation, `cbqsd` acquires one immutable finalized broker-state view. Ownership, provider key, provider status, stream status, escrow availability, authorization generation, encryption generation, pricing schedule, and applicable bounds must all come from that view. Mixing reads from different heights is invalid.

The broker fails closed when chain state is unavailable, internally inconsistent, from another chain instance, or older than the configured staleness bound. These conditions use distinct error codes because their remedies differ: ordinary lag can clear with retry; an unreachable node needs operator action; a regressed or hash-conflicting finalized view must stop service.

Revocation is generation-based. Changing the admin key always increments the authorization generation, and the owner may increment it explicitly. Once a broker observes the finalized change, it terminates or revalidates old sessions before their next mutation. Short grant expiries reduce the exposure between planned rotations.

Encryption and Membership

Record Encryption

Application payloads on an encrypted main stream use XChaCha20-Poly1305 with a random 24-byte nonce. Associated data commits the canonical record header through `encryption_generation`, including `stream_id` and `lane_id`, but not the provider-assigned sequence. Moving ciphertext between lanes therefore fails authentication.

Signing keys and encryption-recipient keys are distinct. v1 uses strict Ed25519 for CBQS signatures and HPKE base mode with X25519-HKDF-SHA256/ChaCha20-Poly1305 for member envelopes. Unknown algorithms or permissive Ed25519 encodings are rejected.

The broker stores ciphertext when clients follow the encryption protocol and sees the cleartext delivery header. It does not receive a generation data key and cannot inspect conforming encrypted application payloads. Provider-generated lane events and key-batch records are cleartext protocol mechanics or HPKE ciphertexts, not stream-encrypted application content.

CBSS Root Escrow

The design places a random 32-byte `stream_root` in CBSS and authorizes one rotation actor to read it through the ordinary job-bound CBSS release path. A conforming rotation workload derives a stream-specific signing seed and generation data keys:

```
rotation_seed =  
    HKDF-SHA256(stream_root, "cbqs/rotation-signing/v1" || stream_id)  
  
data_key[g] =  
    HKDF-SHA256(stream_root, "cbqs/data-key/v1" || stream_id || g_u64_be)
```

CBSS is not on the message path. If it is unavailable, current members can continue using an existing generation, but membership cannot safely rotate. The provider must never invent a fallback key or advance a generation without a verified durable key-batch receipt.

Key Batches

For one encryption generation, a conforming rotation workload HPKE-wraps the 32-byte data key separately to every recipient. A signed `KeyRotationBatchV1` commits the ordered envelope digests, complete membership-set digest, predecessor batch, stream, generation, and signing context.

The broker stages bounded chunks, then atomically materializes one header and the ordered envelopes as contiguous Standard records on the linked key stream. Only after all records, append receipts, receipt tails, batch tail, and `KeyBatchReceiptV1` are durable may the owner activate the generation on chain. This ordering prevents a chain-active generation from stranding retained members without its envelopes.

Adding a member may extend the current generation with a supplemental batch. Removing a member requires a new generation for the retained membership and an atomic authorization-generation bump at activation. Old grants then fail and new appends must name the new encryption generation.

Removal is forward exclusion, not retroactive erasure. A removed member may retain any key or plaintext it previously received. Compromise of the `stream_root` compromises every deterministically derived generation; recovery requires a new root and a new stream, not merely a generation increment.

Receipts and Durability

Receipts are the evidence layer between an off-chain provider and clients that need more than an unsigned HTTP success. Each receipt binds the chain context, stream, provider epoch, one finalized view, a domain-separated receipt kind, and the stream’s snapshotted provider signing key.

Receipt Families

Receipt or anchor	What it commits
<code>AppendReceiptV1</code>	Record identity, header hash, payload hash, generations, global predecessor, and lane predecessor
<code>DeliveryStateReceiptV1</code>	Absolute group or lease result plus prior group receipt
<code>CheckpointReceiptV1</code>	One contiguous durable Fast range and prior checkpoint or epoch transition
<code>BrokerEpochTransitionV1</code>	Clean or void replacement of a Fast lineage
<code>KeyBatchReceiptV1</code>	Durable materialization of one membership batch
<code>StandardRetentionAnchorV1</code>	New retained floor and receipt predecessors needed to restart verification
<code>GroupRetentionAnchorV1</code>	Latest retained group receipt and queue position

Standard appends form one global receipt chain and one predecessor chain per lane. A provider cannot sign two different records at the same global position, or two different continuations of the same predecessor, without producing portable equivocation evidence. Per-lane predecessors let scoped clients verify the history they are entitled to receive.

Group receipts commit absolute post-mutation state rather than only relative commands such as “extend by 30 seconds.” Lease issuance, lease expiry, attempt exhaustion, and record-deadline expiry are signed state transitions, not hidden local bookkeeping. A disconnected verifier can replay the receipt chain to restore continuity.

Fast checkpoints commit an ordered range root and durable high-water mark. Every first checkpoint in a new broker epoch links to the signed epoch transition. This makes a crash-loss suffix explicit rather than allowing a provider to reconnect a client onto an unexplained history.

Canonical Signing

All signed CBQS objects use canonical byte layouts from the shared protocol codec. Integers are unsigned big-endian; fixed-width values are raw bytes; variable bytes and vectors carry explicit bounded lengths; enums use numeric tags; struct fields use the specified order. JSON, CBOR, field names, and transport framing never become signing preimages.

This single-source rule is a security property. If the node, provider, client library, and auditor locally rebuild signing bytes, a difference in field order, optional encoding, integer width, or enum tag can turn an apparently valid receipt into cross-component ambiguity.

Atomic Commit Sets

Except for an explicitly provisional Fast application append, a provider acknowledgement is valid only after the entire operation-specific set crosses one atomic durable transaction:

Operation	Atomic durable set
Standard append	Record, dedupe request digest, exact retry response, append receipt, and global/lane tails
Lane mutation	Lane state, provider event, append receipt, and receipt tails
Group or delivery mutation	Group/lease state, result digest, state update, receipt, replay index, and group tail
Fast batch	Batch records, ordered range root, durable watermark, checkpoint, and checkpoint tail
Broker restart	New broker epoch, clean/void transition, and replacement checkpoint tail before new append
Retention advance	New floor, signed anchor, and pruned records
Key batch	Materialized header/envelopes, append receipts, receipt and batch tails, and key-batch receipt

Recovery replays or rolls back an incomplete transaction as a unit. For a durable operation, a broker must not persist dedupe success before its record, acknowledge a record before its receipt tails, or expose a durable watermark before its checkpoint. The provisional Fast application-append exception does not create a per-record receipt tail.

What Receipts Do Not Prove

A receipt proves what a provider signed. It does not prove that the provider served every authorized consumer, retained data it refuses to return, or remains online. Clients retain their latest append, lane, group, checkpoint, and retention anchors outside the provider when those anchors are needed for audit or migration.

Pricing, Rent, and Stream Lifecycle

CBQS prices reserved capacity instead of provider-reported per-message usage. The chain can therefore settle service without putting message activity on chain or trusting the provider to report bytes honestly.

Two Pricing Paths

Path	Admission authority	Rate behavior
Provider quote	Generation-bound provider signature for one owner, one stream, and exact capacity	Quoted rate is snapshotted into the stream
Governance base	Provider's standing consent and on-chain capacity caps	Deterministic active schedule is snapshotted at creation or capacity update

A provider quote binds the owner, the stream it prices, the stream record's next pricing generation, delivery class, retained bytes, append bytes per second, delivered bytes per second, rate, and expiry. The registry verifies the quote under the provider's current quote key and stores the accepted generation atomically with admission. Single use follows from that binding rather than from a stored marker: a quote names one generation, admission advances it, and a superseded quote can never be replayed.

A quoted rate is a stated price and is never clamped, but it MUST be at or above the protocol rate floor. A zero-rate or token-rate quote is invalid, which removes the path that would otherwise price permanent registry state at nothing.

The base path prices three dimensions for the chosen delivery class:

```
retained_units = ceil_div(retained_bytes_limit, retained_unit_bytes)
throughput_units = ceil_div(max_append_bytes_per_sec,
                             throughput_unit_bytes_per_sec)
delivered_units = ceil_div(max_delivered_bytes_per_sec,
                             delivered_unit_bytes_per_sec)

rate_per_block =
    retained_units * retained_unit_rate_per_block
  + throughput_units * throughput_unit_rate_per_block
  + delivered_units * delivered_unit_rate_per_block
```

The calculation uses checked u128 arithmetic.

Egress is a separate dimension because many consumer groups can read the same record. Charging only retention and ingress would admit fan-out load without reserving the provider's dominant delivery resource.

For base-priced streams, the chain maintains provider reservation totals for stream count, retained bytes, ingress, and egress. A provider may stop new base-rate admissions, but cannot lower a cap below capacity already reserved. A quoted stream does not consume the standing base-rate counters because the provider admitted it explicitly.

Escrow and Settlement

Stream escrow is the native Cowboy token (CBY) held by the registry, not an unbacked accounting field. Creation and top-up move funds into custody. Settlement deducts accrued rent while conserving custody, and distributes each paid amount between the provider and the Platform Fee Account under two governance parameters that MUST sum to 10,000 basis points. The platform share is credited to that account rather than burned: a credited share can still be burned later, a burned one cannot be recovered, and the credit preserves an on-chain record of accrued platform revenue. Close or expiry returns or distributes the remaining balance under the lifecycle rules.

For rate r and effective rent height H :

```
elapsed = H - last_settled_block
due      = elapsed * r
```

If escrow covers **due**, settlement deducts it and advances the paid-through height. Otherwise it advances only by the number of whole blocks the balance can cover, marks the stream **Suspended**, and pins one historical suspension point and grace deadline. Later partial payments do not walk that deadline forward.

A zero initial escrow is valid and creates the stream id, but it does not buy service. Once rent accrues beyond the empty balance, the broker rejects mutations and the chain can materialize suspension.

Creation is nevertheless never free. Every main stream pays a nonrefundable creation charge, debited from the owner in the same atomic transition as the initial escrow and burned rather than split as rent. The charge is priced at the stream's own rate across the grace window a record may occupy while unfunded, so the cheapest possible permanent record still costs real capital. It is deliberately a charge and not an escrow floor: escrow returns to the owner at close, so any floor expressed through escrow is recoverable and prices permanent state at zero. Routing any share of the charge through the rent split would let a self-providing owner recover it. Only a linked key stream carries a zero rate; its lifecycle, escrow, and creation charge are the main stream's.

Pause-Aware Rent Time

Blocks during which the system-actor emergency breaker covers the Stream Registry do not count toward rent or grace deadlines, because the same pause prevents the owner from funding or closing a stream. The registry derives an effective rent height from finalized physical height minus cumulative covered pause blocks. Stored rent heights and broker economic checks use that effective height.

Lifecycle

```
Create -> Active -> Suspended -> Active
                |           |
                +--> DataExpired
```

Active or Suspended -> Closed

Closed and **DataExpired** are absorbing. Suspension preserves the provider's reservation during a bounded funding window. Passing the pinned grace deadline permits the provider to delete data; a later chain settlement materializes **DataExpired** and releases the reservation. Close settles accrued rent, stops new append, and refunds unearned escrow under the control-plane transition.

Provider assignment is immutable. Moving to another provider or a new confidentiality domain means creating a new stream, copying or replaying data through a verified durable anchor, issuing new grants and envelopes, moving clients, and closing the old stream after the target anchor verifies. There is no forced provider reassignment in v1.

Retention and CBFS

The provider stores a bounded hot window. Standard records remain pinned only while applicable groups are nonterminal and inside their configured pin deadlines. Fast retention is independent of client cursors. Capacity pressure returns **Backpressure**; it does not silently evict a live Standard record.

When history moves behind the retained floor, the provider returns **CursorTooOld** together with a signed retention anchor. The anchor identifies the new floor and the predecessor digests needed to restart receipt-chain verification. Floors are monotone. A provider that signs a lower floor after a higher one creates equivocation evidence.

Applications that need colder or provider-independent recovery store snapshots in owner-controlled CBFS volumes. A snapshot manifest binds the stream or lane, snapshot hash, CBFS reference, encryption generation, covered Standard record or Fast checkpoint, provider anchor, and application signature.

A Fast snapshot covers only a durable checkpoint. A snapshotter cannot serialize a live replica that includes provisional operations and call the result durable. Snapshot cadence must beat both broker retention and the application's offline-recovery objective. The broker neither creates nor trusts application snapshots when applying its own retention policy.

Large application payloads use the same division of responsibility. The application encrypts and stores the object in CBFS before appending its encrypted reference in CBQS. CBQS preserves ordering and delivery; CBFS carries the bytes.

Application Patterns

One stream primitive supports several patterns without making payload meaning part of the broker protocol:

Pattern	CBQS construction
Work queue	Standard stream + exact-lane consumer group + leases and terminal states
Fan-out	One Standard replay subscription or one group per independent consumer
Chat room	Standard lane per room; replay cursor for history; attachments in CBFS
Request/reply	Standard records with application correlation and reply routing inside encrypted payloads
CRDT operation log	Fast lane + stable operation ids + checkpoint-aware repair
Presence / typing	Fast records + short application expiry and timeout
Event archive	Standard append receipts + CBFS snapshots beyond hot retention

Homestead

Homestead’s collaborative-document model provides one concrete application pattern. One parent stream represents one uniform cryptographic, provider, retention, and billing domain; one lane represents one page; a lane-set grant covers the pages visible in a member’s sidebar.

Yjs updates use Fast only when the client implements checkpoint-aware repair. Page snapshots are materialized from durable records into CBFS. Search runs client-side or in an explicitly authorized workload that joins the encryption domain and stores its index in a private CBFS volume. Authorizing such a workload expands the confidentiality boundary; the broker never performs semantic search over ciphertext.

The application service remains responsible for identity-to-ACL mapping, grant minting, share links, metadata, seeding, snapshotting, search, and walletless attachment upload. CBQS replaces transport state; it does not become the application database.

Actor Projection

CBQS does not invoke a consensus actor directly. A client helper may consume a stream, derive an application-selected projection, and submit an ordinary actor transaction. That projection

and its timing become public consensus data. The helper must never imply that encrypted CBQS content remains private after it is projected on chain.

Core Protocol

The following sections state the semantic CBQS contract. They intentionally omit field-by-field canonical encodings and exhaustive error registries. In a conforming release, the pinned shared protocol codec is normative for those omitted wire details, and the pinned node handler is normative for omitted chain-transition details. Neither may relax an invariant stated here.

1. Identifiers and Chain Context

1.1. Chain instance

Every signed CBQS object carries both the numeric chain id and a chain-instance id derived from the exact genesis configuration bytes:

```
chain_instance_id =  
    keccak256("cowboy-chain-identity-v1:" || genesis_source_bytes)
```

The instance id separates two chain histories that reuse the same numeric id. The node **MUST** derive it from the same captured bytes parsed for genesis, store it in consensus state, and serve it through finalized broker state. A missing, malformed, or mismatched value fails closed.

cbqsd **MUST** persist the chain-instance binding beside its durable store and refuse startup against a different instance. Reusing application data after a re-genesis requires explicit export and import into new streams with new ids, grants, receipts, and lineages. Changing a configuration value is not a valid rebinding operation.

1.2. Stream and lane identity

The main stream id is owner-nonce-derived and chain-instance-bound:

```
stream_id = keccak256(  
    "cbqs/stream-id/v1" ||  
    chain_id_u64_be ||  
    chain_instance_id ||  
    owner_address_20 ||  
    owner_nonce_u64_be  
)
```

An owner nonce is single-use even after the earlier stream becomes terminal. For an encrypted main stream:

```
key_stream_id = keccak256("cbqs/key-stream-id/v1" || stream_id)
```

Non-default lane ids derive from the parent stream and a client-selected random nonce. The all-zero lane id is reserved as the default lane. Closed lane ids are never reused.

1.3. Provider and broker epochs

Every `v1 provider_epoch` MUST equal zero. Encoders emit zero and verifiers reject any other value. Provider assignment is immutable, so a nonzero epoch cannot be interpreted as reassignment.

Fast streams separately use a provider-controlled `broker_epoch`. The broker durably increments it before accepting records after startup. Record identity therefore remains unique across process restarts even when sequence numbering starts again in the replacement lineage.

1.4. Finalized state

The node exposes one atomic broker-state response containing a `FinalizedViewV1`—chain instance, finalized height, and finalized block hash—together with every registry value needed for the decision and the derived effective rent height. Returning a view and then satisfying follow-up reads from a later state is invalid.

The broker’s observed finalized cursor is monotone. After observing a status, provider-key, provider-epoch, authorization-generation, or encryption-generation change, the broker MUST NOT authorize a later mutation from an older view.

2. Registry Objects and Chain Instructions

2.1. Stream record

The chain-owned stream record contains the following classes of state:

Class	Fields
Identity	version, chain instance, stream id, main/key purpose, owner, owner nonce

Class	Fields
Authority	admin key, rotation key, authorization generation, encryption generation
Assignment	provider, provider epoch, snapshotted provider signing key
Service	delivery class, encryption flag, linked stream id, bounded configuration
Economics	pricing basis, rate per block, escrow, paid-through height
Lifecycle	status, suspension point, pinned grace deadline, creation height

Provider assignment, delivery class, main/key purpose, encryption choice, and linkage are immutable. The owner MAY update permitted configuration fields and keys. Changing the admin key always increments the authorization generation.

An encrypted main/key pair is one lifecycle unit. Creation, update, suspension, reactivation, key activation, close, and expiry MUST transition the pair atomically. The key stream cannot be funded, suspended, migrated, or closed independently.

2.2. Provider record

A provider record contains its public receipt-signing key, price-quote key, bounded TLS WebSocket and HTTPS endpoints, metadata hash, status, live assigned-stream count, standing base-rate admission caps, and registry-maintained reservation totals.

Provider registration is permissionless and creates no slashable stake in v1. An **Active** provider can accept streams. A **Draining** provider serves assigned streams but accepts no new assignments. A provider cannot become **Deregistered** while any live stream remains assigned.

Because registration is permissionless, endpoint URIs are validated at consensus admission under a pinned algorithm: the transport's scheme, a standard URL parse, no userinfo, and a host that is neither the `localhost` namespace nor a literal address outside public unicast — including a host the parser canonicalizes to one. Two conforming implementations MUST accept and reject the identical URI set. Admission cannot resolve DNS, so a dialing client MUST revalidate every resolved address, and every redirect target, against the same address policy before connecting.

Changing a provider signing key is prospective. Existing stream records keep the key they snapshotted, so historical and future receipts for those streams remain in one verifiable key lineage.

2.3. Stream configuration

A configuration bounds retention, retained bytes, maximum message bytes, append and delivered throughput, lane count, and the delivery-class-specific rules. Standard config includes idempotency horizon, visibility limits, and attempt defaults. Fast config includes maximum batch time and record count.

Configuration is validated at creation and update. Some protocol bounds are live admission gates; others are immutable ceilings against which the stream snapshots its own value. A later governance reduction **MUST NOT** silently reinterpret an already accepted stream.

2.4. Chain instructions

CBQS uses top-level system opcode `SYS_CBQS = 218` and one versioned instruction family with nine actions:

Instruction	Authorized caller	Effect
<code>RegisterProvider</code>	Provider account	Create a provider record
<code>UpdateProvider</code>	Provider account	Change endpoints, keys, metadata, or standing admission
<code>SetProviderStatus</code>	Provider account	Move among Active, Draining, and eligible Deregistered states
<code>CreateStream</code>	Owner account	Create and fund a main stream and, when encrypted, its linked key stream
<code>UpdateStream</code>	Owner account	Change allowed config/keys, reprice capacity when required, or revoke grants
<code>TopUpStream</code>	Payer; provider-status rules apply	Move native CBY into stream escrow

Instruction	Authorized caller	Effect
<code>SettleStreamRent</code>	Permissionless	Commit rent accrual and materialize suspension or expiry
<code>CloseStream</code>	Owner account	Settle, stop new append, release reservation, and refund under policy
<code>ActivateKeyGeneration</code>	Owner account	Verify a durable key-batch receipt and advance the active generation

The chain instruction contains no message payload, group, lease, acknowledgement, or cursor. Every handler uses compute-then-commit semantics; failure leaves registry, balances, indices, events, and linked-stream state unchanged.

3. Records, Lanes, and Quotas

3.1. Append body and header

A client append supplies stream and lane ids, a client message id, client timestamp, application expiry, encryption generation, nonce, and bounded payload. The broker validates the request proof, quotas, lane state, and generation, then assigns the record id and canonical header.

Client appends use `origin = Client` and `kind = Application`. Clients MUST set the `v1` delayed-visibility field to zero. Provider lane events use `origin = ProviderEvent`, a lane-event kind, the default lane, an immediately visible header, and a synthetic id derived from the event.

Payload meaning is opaque. Reply routes, semantic topics, application deduplication ids, document operations, and CBFS references belong inside the encrypted payload rather than in broker-visible routing fields.

3.2. Lane behavior

Lane creation and close require `LANE_ADMIN` and consume the parent stream's lane-count and creation-rate bounds. A closed lane rejects new appends but remains replayable while its records remain retained.

Every record carries a cleartext lane id, and the provider maintains an index that can serve an exact-lane subscription without transmitting unrelated records. This is opaque routing-metadata filtering, not payload-semantic filtering.

Multi-lane replay and Fast subscriptions may use Any, Exact, or bounded Set scope. Standard consumer groups remain exact-lane because one cross-lane work queue would blur FIFO and retry order.

3.3. Quota composition

The broker enforces message size, stream retained bytes, ingress, lanes, groups, subscriptions, envelope batches, grant rate, transport credit, and decode-allocation bounds. Delivered capacity is a provider-reservation and pricing dimension at admission. Limits compose by intersection; a grant never widens its stream or provider allocation.

An append that cannot be admitted returns a typed rate or backpressure error. Temporary exhaustion of subscription credit, lease capacity, or local delivery rate leaves delivery pending. The provider **MUST NOT** silently drop a successful Standard append or evict a live Standard record to make room.

4. Authorization and Session Enforcement

4.1. Grant validation

The broker verifies the StreamGrant signature under the admin key read from the operation's finalized view. It then verifies exact context, stream, authorization generation, validity window, requested verb, lane scope, group scope, and every caveat.

`lane_scope = Any` authorizes every lane; `Exact` authorizes one; `Set` authorizes the sorted unique listed lanes. Group scope never widens lane scope. Any or Set on an encrypted stream remains broker-enforced routing isolation because all lanes share the data key.

4.2. Session and request proof

The session proof binds the grant, holder, provider epoch, server challenge, and unique session id. Each post-session request then signs its method and canonical body hash with a strictly increasing counter. Duplicate, backward-skipped, cross-session, or body-mismatched proofs are rejected before mutation.

4.3. In-flight authority

Standard lease issuance requires capacity under both `GroupConfigV1.max_in_flight` and `StreamGrantV1.max_in_flight`. The group counter spans all holders in one group. The holder counter spans every group, subscription, and connection for one holder key on the stream.

The holder count is durable and changes atomically with the lease. Expired leases are released before testing the next issuance so a stale counter cannot permanently wedge a capped holder.

5. Standard Delivery

5.1. Durable append

A Standard append succeeds only after record, idempotency state, original retry response, append receipt, and receipt tails commit atomically. Retrying the exact request within the idempotency horizon returns the original result. Conflicting reuse returns an error and does not append.

The provider stores one monotone sequence per stream and one receipt predecessor per stream and lane. A delivered Standard record includes the matching append receipt; clients reject a header, payload, id, or generation that does not reproduce its committed hashes and identities.

5.2. Group lifecycle

Creating a group selects one lane, a start position, `StrictFifo` or `Concurrent` mode, visibility bounds, an in-flight limit, attempt limit, record deadline, dead-letter lifetime, and optional retention pin deadline. Group ids are never reused, including after deletion.

Each initial cycle receives an immutable increasing group queue sequence. Redrive creates a successor cycle at the current tail with a new sequence. It does not erase the terminal predecessor or rewind the append log.

5.3. Lease transitions

`ACK`, `NACK`, `EXTEND`, and `REJECT` name one exact lease and delivery cycle. A stale lease cannot mutate a later attempt. Every provider-driven lease issue, expiry, attempt exhaustion, and record expiry produces a signed state update in the group's receipt chain.

At-least-once delivery can repeat application effects. Consumers **MUST** make their effects idempotent or transactionally deduplicate in application state. Neither an append receipt nor a consumer `ACK` makes an arbitrary external side effect exactly once.

6. Fast Delivery

6.1. Provisional append

A Fast application-append response carries the current durable watermark but no per-record append receipt. The provider may acknowledge the new record provisionally. A provider-generated lane event is not provisional: it commits durably and carries an append receipt. When either configured batch threshold is reached, the provider stops issuing further provisional acknowledgements for that stream until the batch commits.

The protocol does not publish a fixed latency or throughput multiplier. Batch thresholds bound the honest-storage crash-loss suffix; measured service performance is an operator result, not a constant derived from the protocol.

6.2. Durable checkpoint

Each checkpoint commits one nonempty contiguous record range in global order, its ordered root, prior checkpoint or transition, and durable high-water mark. Missing, duplicated, reordered, wrong-lineage, or noncontiguous records make the checkpoint invalid.

6.3. Restart transition

A Clean transition is valid only when the prior durable and provisional watermarks were equal. A Void transition identifies the last durable checkpoint and invalidates the later provisional suffix. The transition is durable before any replacement-lineage append, and the first replacement checkpoint links to it.

Clients MUST surface and handle a void. An application that cannot reconstruct or resend a lost suffix uses Standard instead.

7. Encryption and Key Activation

7.1. Algorithms and associated data

Encrypted application records use XChaCha20-Poly1305. The associated data binds the record's stream, lane, timing, generation, origin, kind, message id, and nonce. Key envelopes use the fixed v1 HPKE suite. CBQS object signatures use strict Ed25519 verification with fixed key and signature widths.

7.2. Root and generation keys

A conforming rotation workload receives the root only through the owner-authorized, job-bound CBSS release path on its assigned Runner. It **MUST** derive the stream-specific rotation and generation keys, produce member envelopes, and handle root material within the CBSS and Runner secret boundary.

The broker protocol cannot cryptographically force a malicious client to encrypt arbitrary bytes. The confidentiality claim applies when clients use the specified record-encryption and membership protocol; the broker otherwise stores opaque payload bytes.

7.3. Activation ordering

The broker commits a complete signed key batch before the owner submits **ActivateKeyGeneration**. The chain verifies the provider's durable batch receipt and generation context, then advances the active generation. Removal also increments authorization generation so prior grants cannot append or consume under the new membership.

8. Transport, Receipts, and Retention

8.1. Canonical frames

One WebSocket binary message contains one canonical versioned frame. HTTPS long-poll request bodies carry the same client frame and responses carry a bounded batch of the same server frames. Unknown versions, tags, algorithms, oversized fields, invalid UTF-8, semantic identity mismatches, and trailing bytes are decode errors.

8.2. Credit

Open subscriptions have zero credit. A **CreditV1** frame grants record and byte budgets. Sending a record decrements both. The broker limits subscriptions per connection, **MAY** close a persistently uncredited subscription after the slow-consumer timeout, and schedules active subscriptions without starvation.

8.3. Receipt continuity

Standard deliveries include matching append receipts. Group deliveries also include the matching lease-issue state receipt. Fast application deliveries include their current durable watermark

and no Standard lease or per-record append receipt. Provider-generated lane events are durably committed even on a Fast stream and include an append receipt. Clients fail closed when duplicated identity fields disagree.

8.4. Retention floor

Pruning advances a signed monotone anchor atomically with record deletion. `CursorTooOld` carries the relevant anchor so replay can restart at an attested boundary. The broker does not prune Standard records still protected by a valid group pin, and it does not use client snapshots as proof that deletion is safe.

9. Pricing and Lifecycle

9.1. Admission

Creation selects one active provider and either a valid generation-bound quote or the governance base schedule. The accepted rate and pricing-basis digest are stored in the stream. Base-priced admission reserves the requested stream, retention, ingress, and egress capacity under the provider's standing caps.

On either basis the stored rate **MUST** be at or above the protocol rate floor: a base derivation clamps upward to it, and a quote below it is rejected as signed. Creation additionally debits the nonrefundable creation charge, and **MUST** fail without creating the stream when either that debit or the initial escrow cannot be completed.

Changing a priced capacity field settles the prior rate and requires either a new base derivation with checked reservation delta or a fresh provider quote. Unpriced administrative changes do not reprice the stream.

9.2. Rent

Rent accrues on effective, pause-adjusted block height. Settlement uses checked arithmetic and conserves the paid amount. Underfunding pins one suspension point and grace deadline. Top-up checks expiry against pre-deposit economics so an already overdue stream cannot erase its deadline by funding first.

Each paid amount is split between the assigned provider and the Platform Fee Account by two governance parameters that **MUST** sum to 10,000 basis points, enforced on the proposal so neither share is ever the silent complement of a stale partner. The subtraction assigns any indivisible remainder to the platform share, so the two shares conserve the exact paid amount.

9.3. Terminal states

Close and DataExpired release provider capacity exactly once. Both states are absorbing. A rejected transition changes no stream, balance, custody, reservation, or event state. An encrypted main/key pair reaches terminal state atomically.

10. Security Considerations

10.1. Content confidentiality and metadata

On a stream configured for encryption, conforming clients encrypt application payloads end to end. The provider does not receive generation data keys, and the chain receives no payload. The protocol does not prove that arbitrary submitted bytes are ciphertext. This does not provide communication-graph secrecy.

The chain exposes stream existence, owner, provider, configuration, pricing, escrow, status, and generation changes. CBSS exposes secret-policy and rotation-release activity. The provider sees connecting workloads, stream, lane and group identifiers, member envelope recipients, sizes, timing, traffic, cursor, and delivery state. The cleartext lane id can reveal finer workspace topology even when payloads are encrypted.

Provider logs MUST redact payloads, grants, request proofs, key envelopes, and derived keys. Diagnostic identifiers SHOULD use documented truncated hashes that do not include linkable recipient-key material.

10.2. Authorization

A StreamGrant remains powerful until expiry or an authorization-generation bump. Holder keys SHOULD be workload-scoped and excluded from application logs. Brokers MUST verify one finalized view, fail closed on chain-instance mismatch or untrustworthy state, and reauthorize sessions before later mutations after a material finalized change.

Lane and group scopes constrain an honest broker. They do not hide activity from that broker, and lane scope is not cryptographic isolation among holders who possess the parent stream's data key.

10.3. Key management

The stream root, rotation actor, CBSS policy, and assigned rotation Runner are high-value targets. Compromise of the root exposes all deterministically derived generations. A generation

rotation after root compromise is insufficient; the application creates a new root and stream.

Signed key batches prevent broker substitution and make forked membership histories detectable. They do not hide membership from the provider. A removed member retains prior keys and plaintext, so CBQS supplies forward exclusion rather than retroactive revocation.

10.4. Provider failure and equivocation

The assigned provider can censor, delay, or destroy data. v1 has no provider stake, slashing, in-place reassignment, or protocol-level high-availability claim. A provider may replicate its internal store only if every externally visible CBQS rule remains linearizable and receipt-consistent.

Receipt chains turn conflicting signed histories into portable evidence. They do not make unavailable bytes recoverable. Applications with availability requirements **SHOULD** continuously retain a verified CBFS snapshot, another stream anchor, or sufficient client state to migrate.

10.5. Delivery semantics

Standard provides at-least-once lease semantics for records delivered before their terminal deadline while the provider is honest and available. It does not guarantee that every append reaches a consumer before application expiry or a retention-pin deadline. Repeated delivery can repeat an application effect. A consumer must not ACK until it has durably assumed responsibility for that effect.

Strict FIFO can be blocked by a poison record until the consumer rejects it, attempts exhaust, or its deadline expires. Attempt limits, dead-letter review, record deadlines, and controlled redrive are operational safety mechanisms.

10.6. Fast crash loss

Fast may lose every provisional record above the latest checkpoint. The configured batch thresholds bound what an honest implementation continues to acknowledge before forcing a commit, but they are not a cryptographic availability guarantee. Applications without a deterministic or idempotent repair procedure **MUST** use Standard.

10.7. Chain resets and version skew

Chain-instance binding prevents an object from one genesis history being accepted by another history that reused the same numeric chain id. Providers, clients, and nodes fail closed on

unknown mandatory versions, tags, algorithms, frames, receipts, or fields. Components do not synthesize missing state or silently downgrade canonical signing bytes.

A coordinated CBQS release pins the same protocol-codec revision across node, broker, client library, CBSS integration, and every other consumer. Lockfile-local checks are insufficient when different repositories resolve different signing code.

10.8. Denial of service

CBQS bounds message and frame bytes, decode allocations, streams per owner, lanes, groups, subscriptions, in-flight leases, grant lifetime, lane creation, retention pins, envelope batches, ingress, egress, retained bytes, and provider reservations. Arithmetic is checked and capacity failures are atomic.

Permanent registry state is bounded in capital rather than only by counters. A stream record outlives the escrow that funded it, so a count limit alone prices unbounded record creation at the cost of one transaction. The rate floor and the nonrefundable creation charge together make every permanent record cost capital no matter how quickly its stream closes, which is what closes the create-and-abandon growth loop.

The delivery-capacity field is a reserved economic and admission dimension. Runtime delivery is bounded by record and byte credit, connection and subscription limits, and starvation-bounded scheduling. Implementations MAY apply additional local rate shaping without changing the protocol's receipt, ordering, or delivery semantics.

Provider status can deny sponsored top-ups while leaving owner funding and close available. A provider that repeatedly toggles status can waste sponsor transaction fees, but cannot force the owner to retain the stream. v1 exposes that boundary rather than claiming notice-period protection.

10.9. Emergency pause

The Stream Registry participates in the system-actor emergency breaker defined by the Technical Whitepaper. While it is paused, CBQS chain mutations fail before state reads and emit no event, while brokers may continue serving already-authorized traffic from bounded finalized state. Pause-covered blocks are removed from rent and grace clocks so the pause itself cannot expire a stream while all remedies are disabled.

11. Parameters

The following are selected draft v1 defaults. Byte, collection, and time values are validated before allocation. Governance-writable admission bounds apply prospectively; stream-snapshotted configuration is not silently reinterpreted after admission.

11.1. Selected general bounds

Parameter	Default	Meaning
<code>cbqs.max_streams_per_owner</code>	1,024	Live main streams per owner
<code>cbqs.max_provider_endpoints</code>	4	Registered provider endpoints
<code>cbqs.max_inline_message_bytes</code>	262,144	Inline payload bytes (256 KiB)
<code>cbqs.max_wire_frame_bytes</code>	5,242,880	Canonical frame bytes (5 MiB)
<code>cbqs.max_groups_per_stream</code>	4,096	Active Standard groups
<code>cbqs.max_lanes_per_stream</code>	16,384	Lanes including the default lane
<code>cbqs.max_lane_ids_per_grant</code>	4,096	Lane ids in a Set grant
<code>cbqs.max_key_envelopes_per_batch</code>	65,536	Members in one key batch
<code>cbqs.max_lane_creates_per_min</code>	256	Rolling 60-second creates per stream and grant
<code>cbqs.max_lane_list_page</code>	256	Lane enumeration page
<code>cbqs.max_subscriptions_per_connection</code>	256	Active subscriptions per connection
<code>cbqs.max_grant_ttl_ms</code>	86,400,000	Default maximum grant life (24 h)
<code>cbqs.max_clock_skew_ms</code>	60,000	Accepted wall-clock skew
<code>cbqs.max_chain_staleness_blocks</code>	30	Oldest finalized view accepted for service

Parameter	Default	Meaning
<code>cbqs.slow_consumer_timeout_ms</code>	60,000	Zero-credit closure threshold

11.2. Delivery defaults

Parameter	Default	Meaning
<code>cbqs.standard.retention_ms</code>	604,800,000	Standard hot retention (7 days)
<code>cbqs.standard.idempotency_horizon_ms</code>	86,400,000	Standard retry horizon (24 h)
<code>cbqs.standard.visibility_ms</code>	30,000	Default lease visibility
<code>cbqs.standard.max_attempts</code>	10	Default attempt limit
<code>cbqs.fast.retention_ms</code>	3,600,000	Fast hot retention (1 h)
<code>cbqs.fast.batch_max_ms</code>	100	Maximum provisional batch time
<code>cbqs.fast.batch_max_records</code>	1,024	Maximum provisional batch count
<code>cbqs.suspend_grace_blocks</code>	604,800	Effective rent blocks before expiry

The wall-clock duration represented by a block count depends on network block time. Pause-covered blocks do not advance the effective rent height.

11.3. Base-rate schedule

Rates below are nano-CBY per block. Unit sizes and rates are snapshotted into the pricing-basis calculation; they are not throughput measurements.

Class / dimension	Unit	Rate per block
Standard retained	1 GiB	67
Standard ingress	1 MiB/s	13,300
Standard egress	1 MiB/s	66,700
Fast retained	1 GiB	67
Fast ingress	1 MiB/s	8,000
Fast egress	1 MiB/s	66,700

Genesis seeds this schedule and v1 freezes it. The twelve rows are genesis-validation values, not live governance parameters: no chain instruction and no parameter write re-derives the stored schedule. Repricing CBQS to follow a change in the CBFS rate it tracks is a coordinated-release protocol change specified by a later CIP, and that CIP is where a mutable schedule with notice semantics belongs. Until then a CBFS rate increase narrows CBQS provider margins for the life of the schedule — a disclosed, accepted v1 exposure.

11.4. Stream existence

Two constants price the existence of a stream as distinct from its service. Both are compile-time protocol constants with no live governance parameter; retuning them is a coordinated-release change.

Constant	Value	Meaning
Rate floor	8,000 nano-CBY per block	Minimum stored rate on either pricing basis
Creation-charge window	604,800 blocks	Multiplier on the stream's rate for the one-time charge

The charge window equals the genesis suspension grace default, so it is sized to the window an unfunded record may occupy. The charge does not read that governance parameter, so a governance rewrite of the grace window neither cheapens nor breaks creation.

12. Conformance and Availability

This paper defines protocol conformance, not service availability. A component conforms only to the roles and operations it implements; passing a focused codec, node, broker, or client test does not establish an interoperable CBQS service. The Fast provider-event rule in §6.1 remains part of conformance even when a partial broker implements lane mutation only for Standard streams.

A network advertises CBQS only when one coordinated release makes the normative rules in this paper agree with the canonical shared codec and the consensus node implementation. The release **MUST** preserve canonical signing, strict verification, chain-instance isolation, atomic custody and lifecycle transitions, finalized-view authorization, durable Standard acknowledgement, explicit Fast lineage, bounded resource use, and receipt continuity across all participating components.

Deployment packaging, endpoint availability, client encryption behavior, application integration, and measured latency or throughput are separate claims. They require direct evidence and are not implied by protocol conformance.

13. Relationship to the Cowboy Whitepapers

This document layers on, but never supersedes, the Cowboy Technical Whitepaper. Specifically:

- Actors, actor messages, execution, finality, and public state remain governed by the Technical Whitepaper. CBQS never turns an off-chain record into a consensus effect without an ordinary actor transaction.
- Cycles, Cells, account balances, transaction authorization, system actors, governance, and the system-actor emergency breaker follow the Technical Whitepaper. CBQS chain instructions consume those mechanisms normally.
- The Storage Whitepaper governs CBFS volumes used for large encrypted payloads, snapshots, and archived history. A CBFS reference inside a CBQS payload does not change CBFS authorization or durability.
- The Secrets Whitepaper governs CBSS release of the stream root to the authorized rotation workload. CBSS is not a per-message key distributor and does not receive stream payloads.
- Runner assignment and workload execution follow the Technical Whitepaper and Runner architecture. A CBQS StreamGrant authorizes queue operations; it does not authorize a workload to execute or an actor to spend funds.
- Entitlements MAY decide whether an actor or workload can use CBQS at all. StreamGrants are the finer-grained data-plane authority within that allowed capability.

Where this document and the Technical Whitepaper disagree on shared platform values or procedures, the Technical Whitepaper is the normative reference. For details not restated here, the released shared protocol codec is the executable authority for CBQS canonical encodings and signed objects, and the released node handler is the executable authority for chain transitions. A CBQS release is conforming only when those artifacts and this document agree.

End of specification.