Sibuna: Foundation Architecture, Delivery Plan, and Performance Contract

STATE
published
TYPE
Architectural Specification
CREATED
2026-09-07
STATUS
Published
AUTHORS
Sibuna Contributors <team@sibuna.local>
UPDATED
2026-10-15

SUMMARY
Foundational architectural specification, zero-allocation pipeline, two-tier proof-of-work engine, keyed-hash session tokens, product surfaces, measured performance contract, and delivery record for the Sibuna pure-Zig monorepo

DISCUSSION
No external discussion URL assigned

LABELS
architecture, firewall, pow, performance

Record status

The catalog defines the review process and the meaning of each state. This numbered record forms part of the catalog.

Contents

1. Revision note 1
2. Abstract 3
3. Decision 4
4. Request contract 4
5. Authentication and work 5
6. Policy and memory 5
7. Measurement gates 6
8. Remaining limits 6
9. References 6

1. Revision note

2026-10-15: Ported from the Typst original in the Zig repository. The record describes the Zig implementation as it was decided; SID 0010 maps its packages to the Rust crates of the port.

2026-10-15 (Rust port): the reverse proxy (`sibuna-net`) changes three mechanisms of the request contract. First, the origin pool still holds at most 256 idle sockets, but in 16 stripes of 16 sockets with a lock each; a connection thread gives to and takes from its home stripe first and looks into the other stripes before it connects afresh. Zig kept one pool under one spin lock, which every proxied request contended for. Second, an accepted WebSocket upgrade is relayed by two threads: the connection's own thread carries origin-to-client bytes, and a scoped thread with a 128 KiB stack carries client-to-origin bytes. Zig served both

directions from one thread with `poll(2)`, which the Rust standard library does not expose; two threads with socket timeouts keep the relay free of `unsafe`. An upgraded connection therefore holds two threads, so the number of connection threads can reach twice `--max-connections` when every connection is upgraded. Half closes, failure handling and the idle bound over both directions are unchanged. Third, the proxy borrows the parsed request instead of re-reading the connection buffer, and reads the rest of a streamed upload into a per-connection upload buffer (SID 0009). Bytes on the wire, the retry rule and the keep-alive decisions are unchanged; the port replays scenarios recorded from the Zig proxy and compares what the origin and the client receive.

2026-10-15 (Rust port): the policy engine is no longer a fixed-capacity value held in two double-buffered slots with an arena per slot and a reader count. `sibuna-policy` builds an owned `Engine` off the request path (about 200 KiB of compact heap tables, reported by `Engine::heap_bytes`, where the Zig value was 853,728 bytes), and the daemon publishes it behind an `Arc`. Each request thread reads it through a cache that checks a generation counter with one acquire load and clones the `Arc` only after a publication; an old engine is freed when the last thread holding it moves on, and no writer ever waits for readers to drain. The reason is ownership: reference counting replaces the slot protocol and its arena reset rules, so an engine cannot be freed or rebuilt while a request still reads it. The capacities (128 rules, 128-byte rule names, 8,192 trie nodes) and every decision are unchanged. The signature automata use byte classes instead of 256 columns per state (SID 0006, Part F). Storage is SQLite or PostgreSQL through sqlx (SID 0011) instead of Zaxonlite.

2026-10-15 (security review, Rust port): the reverse proxy and the accept path gain five protections; they change behaviour only towards faulty origins and hostile clients. First, an origin socket returns to the pool only at a clean message boundary: the origin keeps the connection, the body was framed by `Content-Length` or chunked coding and read to its end, and no byte follows it in the read buffer. Bodiless responses (HEAD, 204, 304) and exchanges whose request carried a body are never pooled, because an origin that sends a body after a bodiless reply, or that ignores a body and answers it later as a request of its own (CL.0), would hand that answer to the next client. Taking a socket from the pool first peeks at it without blocking; a socket with bytes waiting or at end of stream is closed. This costs about 1 us per pooled exchange and one fresh origin connection per request with a body. Second, `--max-connections-per-ip <n>` (default 64, 0 disables) bounds the connections one socket peer address may hold; an IPv4-mapped IPv6 address counts with its IPv4 address, and forwarded headers play no part. Beyond the bound the accept loop answers `503` and closes the socket as for `--max-connections`, counting the event in `sibuna_overloaded_total`. The counts live in a fixed keyed-hash table of 1,024 buckets of 16 clients. Third, progress is required in both directions: the client socket carries a write timeout equal to `--idle-timeout`, so a client that accepts no byte for that long loses the connection whatever the origin sends, and an upgraded connection ends when no byte moves either way for `--websocket-idle-timeout` or when either peer accepts no byte for that long, so traffic one way no longer keeps a peer that stopped reading alive. (Corrected 2026-10-15: an earlier wording required each direction to deliver a byte on its own, which would end legitimate one-way streams; the relay never did that.) A client that never reads can still occupy the slack of the socket buffers, which the standard library gives no way to shrink. Fourth, every response Sibuna writes to a HEAD request, its own pages included, carries the head, with the `Content-Length` of the GET response, and no body. Fifth, a request field whose name equals a field Sibuna sets or strips with _ written for - (`X_Forwarded_For`) is dropped, a request with more than one `Host` field is answered `400` (RFC 9112 section 3.2), and a `Connection` field cannot nominate `Host`. The origin's `Content-Length` and chunk sizes must be plain digits (no sign, underscore or surrounding junk), chunk data must end in CRLF, and trailer lines must be field lines; anything else is a malformed upstream response: `502` before the head is relayed, an aborted exchange after it. Zig accepted `+3` and `1_0` through `parseInt` and relayed them as sent.

2026-10-15 (Rust port, security fix): with forwarded identity trusted, the client is the `X-Forwarded-For` entry `--trusted-hops` positions from the right (default 1, the address the nearest trusted proxy appended), reading all `X-Forwarded-For` fields as one list; `X-Real-IP` is read only without `X-Forwarded-For`. An entry that is not an IP address is not trusted: the request is judged by the socket peer and counted (`sibuna_forwarded_invalid_total`). Zig took the leftmost entry, which appending ingresses pass through from the client, so a client could choose its own identity: ban others, reuse one solved proof across addresses and evade every per-client limit. The address is parsed once per request.

2026-10-15 (Rust port): the origin receives `X-Forwarded-For` as a reverse proxy passes it on: the received chain when forwarded identity is trusted, followed by `, `` and the socket peer; without trust, the socket peer alone, never a client-written entry. ``X-Real-IP` is the client the request was judged by. Zig wrote the judged client into both fields. Origins behind Sibuna must read the client from `X-Real-IP` or from the right of `X-Forwarded-For`, never from its leftmost entry.
2026-10-15 (Rust port, security fix): `--max-connections-per-ip` counts an IPv6 socket peer with its `--ipv6-prefix` network (a /64 by default), as limits and bans do, so a client holding a /64 can no longer open the allowed connections from each of its addresses. IPv4 addresses are counted one by one, as before.
2026-10-15 (Rust port, proxy bounds): `sibuna-net` adds three defensive bounds. The proxy sends at most the declared `Content-Length` of a request body, even when more bytes follow it. The search for the end of an origin's response head resumes where the previous search stopped, so a head that arrives byte by byte costs linear work. An origin chunk size near 2^{64} no longer overflows the arithmetic of the relay. Shutdown reaches active origin sockets through ownership: each exchange holds a reference-counted handle to its origin socket, and the idle table holds a duplicate of each client socket, so a socket cannot close, and its descriptor cannot be reused, while the reaper can still reach it. Zig kept every active origin socket in a table of 8,192 entries under the pool lock for the same purpose. Idle stamps are milliseconds of a monotonic clock that starts at its first use in the process, instead of Zig's awake clock.
2026-10-15 (Rust port, daemon): a request may carry several values of the session cookie, in one `Cookie` field or in several. The daemon tries them in order, at most eight, and the first that verifies and carries enough work satisfies the challenge; each value tried costs one verification. Zig read only the first value, so a page on a sibling domain could plant a cookie of the same name ahead of the real one (cookie tossing) and keep its visitors in a challenge loop. SIGINT and SIGTERM are awaited with `sigwait` on a dedicated thread, which wakes the main thread at once; Zig's signal handler set a flag that the main thread polled every 100 ms. An accept error other than an interrupted call pauses the accept thread for 1 ms instead of retrying at once, so running out of descriptors does not spin a CPU. Request references on HTML pages come from a per-thread xorshift generator instead of the system random source; they are references, not secrets.
2026-10-15 (Rust port, performance): connection threads are reused. Zig started a thread for every accepted connection and ended it with the connection; the port did the same at first. Creating and ending a thread maps and unmaps its signal stack and returns its stack pages, and those address-space changes stalled the daemon's other threads: with 64 connections each closed after 4,096 requests, the 99.9th percentile at 100,000 admitted requests per second was 0.3 to 2 ms, against 0.1 ms for Zig. An experiment that never closed connections (so no thread was created during the run) gave 54 to 60 us, which isolated thread creation as the cause. A thread that finishes its connection now parks and takes the next one; a thread parked for 10 seconds ends and is joined by the next launch or the next thread to finish a task, so a burst does not leave its stacks resident. A thread whose connection panics ends instead of parking. At most `--max-connections` connections are served at once, as before. The 99.9th percentile at 100,000 requests per second is now 62 us (Zig 112 us) in forward-auth mode and 150 us (Zig 1,597 us) in reverse-proxy mode (`benchmarks/results/latency-latest.md`).
2026-10-15 (Rust port, bug fix): a connection gives its place under `--max-connections` and `--max-connections-per-ip` back before it writes the response that ends it (`Connection: close`, or the last of the 4,096 requests a connection may carry), instead of after its thread returns. A client may reconnect as soon as it reads that response; when the old place was still counted, a client at its per-client bound was refused with `503`. The whole-product benchmark showed it as a few unexpected answers in two million (`sibuna_overloaded_total`), and a regression test reproduces it under CPU pressure (`cmd/sibuna/tests/hardening.rs`).

2. Abstract

Foundational architectural specification, zero-allocation pipeline, two-tier proof-of-work engine, keyed-hash session tokens, product surfaces, measured performance contract, and delivery record for the Sibuna pure-Zig monorepo.

Implementation review (2026-09-07)
This revision follows code review and regression tests. Performance results live in `benchmarks/results/latest.json` and `distributed-latest.json`; earlier fixed latency tables and unsupported competitor cost models have been removed. Measurements are host-specific, not proofs of optimality or deployment capacity.

Whole-product measurement (2026-09-07, later the same day)
Driving the daemon with an external load generator (`benchmarks/tools.py`, results in `tools-comparison-latest.json`) exposed three limits invisible to the primitive suite: accept threads served one connection to completion (tail latency over 100 ms at 64 clients), every proxied response closed the client connection, and every proxied request opened a new origin connection (about 1,400 requests per second before ephemeral-port exhaustion). The request contract below now describes bounded per-connection threads with a `503` overload path, origin response framing, and a pooled origin connection with one safe retry.

3. Decision

Sibuna provides two product surfaces from one Zig binary. Persistent storage and distributed edge deployment are options for these surfaces, not a third product.

Capability	Gate (`--gate`)	Shield (`--shield`, default)
Admission	Hashcash or sequential-work browser challenge; bound session token	Same
Policy	Declarative rules, bot signatures, CIDR reputation	Same
Flood controls	GCRA, honeypot, temporary bans	Same
Payload inspection	Disabled	SQL injection, XSS, traversal, command-injection heuristics
Deployment	Reverse proxy or forward auth	Reverse proxy or forward auth
Optional state	Embedded persistent policy and reputation	Also WAF incident forensics
Distributed deployment	Cluster build and configured members	Same

4. Request contract

The server runs bounded blocking accept threads (`--workers`, one per CPU by default). Each accepted connection is served on its own thread with a one-megabyte stack, so a slow or idle peer never delays another connection; the number of connection threads is bounded by `--max-connections` (1,024 by default), beyond which the accept loop answers `503` and closes the socket without spawning, counting the event in `sibuna_overloaded_total`. A finished connection thread still owns its stack until it is joined, and it is joined when its slot is next taken, so slots are taken lowest first and the number of allocated stacks follows the connections currently open rather than the number served. The idle reaper closes connections silent for longer than `--idle-timeout`. Each active connection owns a 64 KB read buffer, with a 16 KB head limit, and bounded write buffers. HTTP/1.0 and HTTP/1.1 are supported. Chunked request bodies are decoded in place and re-framed for the origin (SID 0009); other transfer codings are refused with 501. Invalid header names, request-line control bytes, conflicting lengths, and invalid decimal lengths are refused. Truncated bodies fail closed. Requests whose declared body exceeds the buffered prefix do not reuse the client connection after a local response.

The response path applies these checks in order:

1. Local ban table.
2. Reserved internal routes (assets, challenge issuance, verification, honeypot, health, metrics).

3. Local GCRA admission.
4. Active policy snapshot: Shield inspection, reputation, ordered rules, scoring, static paths, bot signatures and default admission.
5. A valid session satisfies an admission challenge; it never overrides a denial or WAF finding.
6. An admitted reverse-proxy request streams to the origin; forward auth returns a verdict. The proxy reads the origin's response head (at most 16 KB), classifies the body framing by RFC 9112 (no body, `Content-Length`, chunked, or close-delimited), relays it exactly, and keeps the client connection open unless the response was close-delimited. Origin sockets left at a clean boundary by a persistent response return to a fixed pool of 256; a pooled socket the origin has closed is retried once on a fresh connection before any byte reaches the client, and only when the request body was fully buffered.

Rust port note (2026-10-15)
The pool of 256 is striped (16 stripes of 16 sockets, a lock each), and an upgraded WebSocket connection uses a second relay thread. See the revision note.

The server's primitive parsing, evaluation and verification routines take no allocator. This is a source/API contract, not a claim that the networking runtime, thread creation, startup, or persistent storage perform no allocation. Benchmark instrumentation is linked only into the standalone benchmark executable. Production metrics and snapshot reader counts still have real atomic costs; they are not benchmark overhead.

5. Authentication and work

A 32-byte master seed derives independent token, challenge, fingerprint and signing keys. The normal token is a 40-byte payload (version, the work level the holder solved, timestamp, expiry, rule hash, fingerprint) plus a 16-byte keyed BLAKE3 tag, URL-safe base64 encoded; a session admits only routes whose demanded work level it reaches (amended 2026-10-01). Ed25519 is an optional asymmetric token format. Tokens are bound to client address and User-Agent and expire. Trust forwarded identity only behind an ingress that overwrites it.

Issuing a challenge stores no entry. Successful verification inserts the authenticated tag into a 16-shard spent set. Failed bounded insertions leave every live tag intact and refuse admission. The set is per process and disappears on restart. In cluster mode, challenge keys are additionally derived from the node id at startup, while token keys remain shared. Challenge issuance and solution submission must reach the same node; the resulting token works on every member with the same seed. Distinct cluster node ids are required. This avoids cross-node replay without a request-path consensus transaction. It does not make replay state durable across restarts.

Tier 1 verifies `SHA256(challenge || ":" || decimal(nonce))` with bit-level difficulty. Tier 2 labels the sequential-work graph in `libs/crypto/src/posw.zig`; both native verification and the WASM prover import the same source. SID 0006 separates graph/hash evaluations from SHA-256 compression counts and states the research assumptions. Tests against a reference labeller are correctness evidence, not an independent cryptographic audit.

6. Policy and memory

The engine owns fixed-capacity rules, a radix trie, and dense Aho-Corasick tables. Signature capacity is one plus the sum of pattern lengths: no trie can require more states. This retains one transition lookup per input byte while avoiding the former oversized tables. Exact engine and table sizes are emitted by the benchmark using `@sizeOf`.

Dynamic storage owns two engine slots and an arena dedicated to each slot. A slot's arena is reset only when that slot is inactive and prior readers have drained. Response rule names are copied into a bounded 128-byte buffer, then the reader releases its slot before any proxy I/O. Pointer publication and reader-counter operations use sequential consistency so the cross-atomic ordering argument is valid. Storage polling, SQL, vector search and persistence stay on the storage thread.

Rust port note (2026-10-15)
The Rust daemon has no engine slots, slot arenas or reader counts: it publishes an owned engine behind an `Arc`, and sizes come from `Engine::heap_bytes` instead of `@sizeOf`. See the revision note.

7. Measurement gates

```
zig build fmt
zig build test
zig build test -Dstorage=false
zig build test -Dcluster=true
sh benchmarks/run-all.sh
python3 benchmarks/distributed.py
zig build sid
```

The primitive harness warms inputs, resets mutable stores outside the timer before each of seven batches, and prevents pure work from being hoisted out of loops. No operation reads a clock. Its bot implementations search identical pattern sets with equivalent any-match semantics. Allocation values are source-audited expectations, not allocator measurements.
The distributed harness starts three actual daemon processes and six external client processes. It checks status codes, sessions across nodes, authenticated WAF denial, issuer-bound challenge rejection, reputation propagation, and continued HTTP service with one member down. Batch timings include client and loopback costs. The harness tests loopback PSK and mutual TLS. WAN behavior needs separate measurements.

8. Remaining limits

Shield is a bounded heuristic inspector, not a full SQL/HTML parser. Only the first 8 KB of body is inspected, and encoded fields longer than the canonical buffer are inspected raw. There is no HTTP/2, native TLS termination, global rate quota, durable replay set, administrative UI, or measured volumetric network mitigation. Forward-auth inspection sees only bytes the ingress sends. Blocking workers and slow origins limit concurrency. No claim of full managed edge-service equivalence follows from a local benchmark.

9. References

- SID 0006, *Mathematical Foundations of Sibuna: Sequential Work, Symmetric Authentication, Bounded State, and Linear-Time Inspection*, which separates graph/hash evaluations from SHA-256 compression counts and states the research assumptions.
- SID 0009, *Chunked Request Bodies: Strict In-Place Decoding, Canonical Re-Framing, and Inspection Equivalence*, which specifies the decoding and re-framing of chunked request bodies.
- RFC 9112, HTTP/1.1, which defines the message body framing the proxy classifies.
- `benchmarks/results/latest.json`, `distributed-latest.json` and `tools-comparison-latest.json`, the measured results in the Zig repository.

Lifecycle event

```
2026-10-15: prediscussion -> discussion. Ported from the Zig repository's
Typst record
```

Lifecycle event

```
2026-10-15: discussion -> published. State carried over from the Zig
repository's registry
```