ProbeChain V2.0 Technical Whitepaper

Version 2.0 — March 2026

Abstract

ProbeChain is a next-generation blockchain platform engineered for sub-second finality, verifiable validator behavior, post-quantum cryptographic security, and agent-first smart contract programmability. Built on **Proof-of-Behavior (PoB)** consensus with **400-millisecond block production** (StellarSpeed), ProbeChain introduces a five-dimension behavioral scoring system that replaces energy-intensive mining with meritocratic validator selection. The platform features the **PROBE Language**, an agent-first programming language with linear types, a register-based virtual machine, and native post-quantum cryptography opcodes. ProbeChain's **Stellar-Class Resilience** layer extends consensus to off-grid and interstellar scenarios through Rydberg atomic receiver technology and radio-frequency block propagation. This whitepaper provides a comprehensive technical specification of the ProbeChain V2.0 architecture.

Table of Contents

1. Introduction

The blockchain ecosystem faces several fundamental challenges: high energy consumption from Proof-of-Work, slow finality times measured in minutes, vulnerability to quantum computing advances, and smart contract languages that are not designed for the emerging AI agent economy. ProbeChain V2.0 addresses each of these challenges through a purpose-built architecture.

1.1 Design Principles

- **Behavior over resources.** Consensus influence derives from verifiable trustworthy conduct, not computational power or capital stake.
- **Sub-second finality.** 400ms block production enables real-time applications.
- **Quantum readiness.** Native Dilithium (ML-DSA) signatures and post-quantum verification opcodes ensure forward security.
- **Agent-first programmability.** The PROBE Language treats AI agents as first-class citizens with native `spawn`, `send`, and `recv` primitives.
- **Signal sovereignty.** The Stellar RF layer enables consensus participation independent of internet infrastructure.

1.2 Network Parameters

Parameter	Value
Chain ID	142857
Block Time	400 ms (StellarSpeed)
Consensus	Proof-of-Behavior (PoB)
Token	PROBE
Decimals	18
Total Supply	10,000,000,000 PROBE
Smallest Unit	Pico (1 PROBE = 10^18 Pico)
Epoch Length	30,000 blocks
Validator Reward	1 PROBE per block

2. PROBE Token Economics

2.1 Denomination Table

Unit	Value in Pico	Relation
Pico	1	Base unit
GPico	10^9	1 billion Pico
PROBE (Probeer)	10^18	1 quintillion Pico

The PROBE token uses 18 decimal places, providing sufficient granularity for micropayments and gas accounting. The total supply of 10 billion PROBE is fixed at genesis.

2.2 Genesis Allocation

The genesis block allocates the full 10,000,000,000 PROBE (10^28 Pico) across:

- **Validator staking reserves** — Initial validator set collateral
- **Ecosystem development** — Protocol development and tooling
- **Community allocation** — Grants, airdrops, and participation incentives

2.3 Block Rewards

Recipient	Reward
PoB Validator	1 PROBE per block
PoW Miner (transition period)	3 PROBE per block
Uncle inclusion (transition)	0.5 PROBE

Block rewards are the sole source of new token issuance beyond the genesis allocation. The PoB validator reward of 1 PROBE per block produces approximately 78,840,000 PROBE per year at 400ms block intervals, representing an annual inflation rate under 0.8%.

3. Proof-of-Behavior Consensus

3.1 Overview

Proof-of-Behavior (PoB) is a novel consensus mechanism that links a validator's block production rights to its verifiable on-chain conduct. Unlike Proof-of-Stake, which ties influence to capital, or Proof-of-Work, which ties influence to computational resources, PoB ties influence to **demonstrated trustworthiness** as measured by an AI-powered behavioral scoring agent.

PoB rests on three pillars:

1. **Layered Utility Scoring** — An AI Behavior Agent scores each validator across five behavioral dimensions.
2. **Dynamic Weight Adaptation** — A validator's probability of producing the next block scales with its behavior score.
3. **Proportional Slashing** — Misbehavior incurs score penalties proportional to both severity and current standing.

3.2 Configuration

```
Period:             0 (StellarSpeed mode: 400ms ticks)
Epoch Length:       30,000 blocks (~3.3 hours at 400ms)
Initial Score:      5,000 basis points (out of 10,000)
Slash Fraction:     1,000 basis points (10%)
Demotion Threshold: 1,000 basis points
Checkpoint Interval: 1,024 blocks
Signature Cache:    4,096 entries
```

3.3 The Five-Dimension Behavior Scoring System

The BehaviorAgent evaluates each validator across five orthogonal dimensions, each scored from 0 to 10,000 basis points:

3.3.1 Liveness (Weight: 25%)

Measures a validator's reliability in producing blocks when scheduled.

```
Liveness = (BlocksProposed × 10,000) / (BlocksProposed + BlocksMissed)
```

A validator that never misses a block slot receives the maximum score of 10,000. Each missed block opportunity reduces the score proportionally.

3.3.2 Correctness (Weight: 25%)

Measures the validity of a validator's block proposals.

```
Correctness = (BlocksProposed × 10,000) / (BlocksProposed + InvalidProposals)
```

Invalid proposals — blocks that fail header validation, contain malformed transactions, or violate protocol rules — reduce the correctness score.

3.3.3 Cooperation (Weight: 18%)

Measures participation in the consensus acknowledgment (ACK) protocol.

```
Cooperation = (AcksGiven × 10,000) / (AcksGiven + AcksMissed)
```

Validators are expected to acknowledge blocks produced by other validators. Consistent ACK participation strengthens network consensus and earns a higher cooperation score.

3.3.4 Consistency (Weight: 17%)

Measures freedom from slashing events.

```
Consistency = max(0, 10,000 − SlashCount × 1,000)
```

Each slashing event deducts 1,000 basis points (10% of the maximum score). A validator with 10 or more slashing events receives a consistency score of zero.

3.3.5 Signal Sovereignty (Weight: 15%)

Measures a validator's Stellar-Class capabilities — its ability to maintain accurate timekeeping through independent physical references.

For validators without any Stellar-Class operations, the score defaults to 5,000 (neutral baseline). For active stellar validators, three components are evaluated:

- **Rydberg Verification (40%):** Proportion of stellar blocks with Rydberg atomic clock verification
- **Radio Synchronization (30%):** Number of successful RF-based time syncs (each adds 100 bp, capped at 10,000)
- **Stellar Block Production (30%):** Number of blocks produced with AtomicTime field (each adds 50 bp, capped at 10,000)

3.4 Composite Score Calculation

The composite score is the weighted sum of all five dimensions:

```
Total = (Liveness × 25 + Correctness × 25 + Cooperation × 18
       + Consistency × 17 + SignalSovereignty × 15) / 100
```

The total is capped at 10,000 basis points.

3.5 Proportional Slashing

When a validator is slashed, the penalty is proportional to both the severity of the infraction and the validator's current score:

```
Deduction = (CurrentScore × Severity × SlashFraction) / (10,000 × 10,000)
NewScore = CurrentScore − Deduction
```

This design means that high-scoring validators lose more in absolute terms, creating a stronger deterrent for well-established validators to engage in misbehavior. Validators with already-low scores face smaller absolute penalties but are closer to the demotion threshold.

3.6 Validator Selection

Block production rights are assigned based on behavior scores:

1. The validator must be in the active validator set.
2. The validator's score must exceed the demotion threshold (1,000 bp).
3. Higher-scoring validators receive more frequent block production opportunities.
4. A cooldown mechanism prevents consecutive block production by the same validator.

In-turn blocks (produced by the expected validator) carry difficulty 2; **out-of-turn** blocks carry difficulty 1, ensuring the chain favors orderly block production.

3.7 Validator Lifecycle

Event	Mechanism
Admission	Majority vote (>50%) by current validators
Initial Score	5,000 basis points
Promotion	Score rises through consistent behavior
Demotion	Automatic when score falls below 1,000 bp
Removal	Majority vote by current validators

3.8 ACK Protocol

Validators broadcast acknowledgments for blocks they observe:

```
ACK = {
    EpochPosition: uint8,    // Position in current epoch
    Number:        uint256,  // Block number
    BlockHash:     bytes32,  // Hash of acknowledged block
    AckType:       uint8,    // 0=Agree, 1=Oppose, 255=All
    WitnessSig:    bytes,    // ECDSA or Dilithium signature
}
```

ACKs serve dual purposes: they accelerate consensus convergence and contribute to cooperation scoring.

3.9 Epoch-Boundary Evaluation

Full behavioral re-evaluation occurs every 30,000 blocks (~3.3 hours). Between epochs, the BehaviorAgent returns cached scores with updated timestamps, enabling StellarSpeed's 400ms tick interval without computational overhead from continuous re-scoring.

---

4. StellarSpeed: Sub-Second Block Production

4.1 Architecture

StellarSpeed is ProbeChain's sub-second block production engine. When active (the default since genesis), blocks are produced at 400-millisecond intervals through a pipelined validation architecture.

```
StellarSpeed Configuration:
  Tick Interval:      400 ms
  Pipeline:           Enabled (next block prep begins immediately after seal)
  Reduced ACK Quorum: Enabled (quorum = normalQuorum / 2, minimum 1)
  Future Block Limit: 1 second (reduced from 15 seconds)
```

4.2 Pipelined Block Production

In StellarSpeed mode, block production is pipelined:

1. **Tick 0:** Validator V1 seals Block N
2. **Tick 0+ε:** V2 begins assembling Block N+1 immediately
3. **Tick 400ms:** V2 seals Block N+1, V3 begins Block N+2

This pipeline eliminates idle time between block production cycles and sustains the 400ms cadence.

4.3 Reduced ACK Quorum

To maintain sub-second finality, StellarSpeed halves the ACK quorum requirement:

```
ReducedQuorum = max(1, NormalQuorum / 2)
```

Fewer acknowledgments are needed per block, trading marginal consensus strength for throughput. The behavioral scoring system compensates by tracking ACK participation and penalizing non-cooperative validators.

4.4 Same-Second Block Ordering

At 400ms intervals, multiple blocks may share the same Unix timestamp (which has 1-second granularity). ProbeChain resolves ordering through the AtomicTime header field, which provides nanosecond precision. Blocks with identical Unix timestamps are ordered by their AtomicTime nanoseconds.

4.5 Performance Characteristics

Metric	Value
Block interval	400 ms
Blocks per minute	150
Blocks per hour	9,000
Blocks per day	216,000
Blocks per epoch (30,000)	~3.3 hours
Transaction confirmation	< 1 second

---

5. Stellar-Class Resilience

5.1 AtomicTime

AtomicTime is a 17-byte timestamp embedded in every block header that provides nanosecond-precision time ordering with explicit clock source metadata and uncertainty bounds.

```
AtomicTimestamp {
    Seconds:     uint64  [8 bytes]  // Unix seconds (TAI-based)
    Nanoseconds: uint32  [4 bytes]  // Sub-second precision
    ClockSource: uint8   [1 byte]   // Time sync source
    Uncertainty: uint32  [4 bytes]  // Estimated uncertainty (nanoseconds)
}
```

Clock Sources and Typical Uncertainty

Source	ID	Uncertainty	Technology
Rydberg	4	~1 ns	Rydberg atomic clock
GNSS	3	~100 ns	GPS / Galileo / GLONASS
PTP	2	~1 μs	IEEE 1588 Precision Time Protocol
NTP	1	~10 ms	Network Time Protocol
System	0	~100 ms	OS system clock

5.2 Uncertainty Overlap

Two timestamps are considered **temporally coincident** when their uncertainty intervals overlap:

```
|DurationBetween(T1, T2)| ≤ T1.Uncertainty + T2.Uncertainty
```

This enables graceful degradation: nodes with low-precision clocks can still participate, while nodes with Rydberg references can establish tighter ordering.

5.3 Light Propagation Delay

For interstellar network participants, ProbeChain computes one-way light propagation delays:

```
c = 299,792,458 m/s
1 AU delay  ≈ 500 seconds
1 ly delay  ≈ 31.6 billion seconds
```

These delays are factored into timestamp validation for deep-space nodes, ensuring that block ordering remains physically consistent across solar-system-scale distances.

5.4 Stellar RF Layer

The Stellar RF layer enables block propagation over radio frequency, bypassing internet infrastructure entirely. This is critical for:

- **Off-grid mining operations** in remote locations
- **Disaster recovery** when internet connectivity is lost
- **Interplanetary networks** where TCP/IP latency is impractical

5.4.1 RadioEncap Interface

Block data is encoded for RF transmission through the RadioEncap interface:

```
RadioEncap {
    EncodeBlock(block) → RF stream    // RLP → FEC → Frame
    DecodeBlock(stream) → block       // Frame → FEC → RLP
    BandInfo() → BandDescriptor
    SyncPreamble() → bytes
}
```

The encoding pipeline applies Reed-Solomon forward error correction (16 parity shards default, 32 for noisy HF band), followed by framing with a sync preamble (0xAA 0x55 0xAA 0x55 "PRBE"), length header, and CRC32 integrity check.

5.4.2 Supported RF Bands

Band	Frequency Range	Channel BW	Modulation	Data Rate	Use Case
HF	3–30 MHz	3 kHz	BPSK	1.5 kbps	Long-range skywave
VHF	30–300 MHz	25 kHz	QPSK	25 kbps	Line-of-sight
UHF	300 MHz–3 GHz	200 kHz	QPSK	200 kbps	Urban/suburban
SHF	3–30 GHz	1 MHz	8PSK	1.5 Mbps	Satellite links
EHF	30–300 GHz	10 MHz	16QAM	20 Mbps	Millimeter wave
THz	300 GHz–3 THz	100 MHz	BPSK	50 Mbps	Experimental Rydberg

Data rates include a 50% efficiency factor for FEC and framing overhead.

5.4.3 Rydberg Atomic Receivers

Rydberg atomic receivers exploit quantum transitions in Rydberg atoms to achieve extraordinary RF sensitivity:

```
Default Rydberg Capability:
  Bands:       HF through EHF (5 bands)
  Sensitivity: -140 dBm
  Atomic Ref:  Yes (built-in atomic clock)
  Data Rate:   1 Mbps baseline

Full Spectrum (with THz):
  Bands:       HF through THz (6 bands)
  Sensitivity: -150 dBm
  Atomic Ref:  Yes
  Data Rate:   10 Mbps
```

Validators equipped with Rydberg receivers earn Signal Sovereignty score bonuses, incentivizing physical decentralization of the network's time reference infrastructure.

---

6. Post-Quantum Cryptography

6.1 Dilithium Integration

ProbeChain integrates CRYSTALS-Dilithium (ML-DSA-44) as a drop-in replacement for ECDSA at the consensus layer.

Property	ECDSA	Dilithium (ML-DSA-44)
Public Key	33 bytes	1,312 bytes
Signature	65 bytes	2,420 bytes
Security Level	128-bit classical	NIST Level 2 (quantum-safe)
Signature Type Detection	Length-based	3,732 bytes = pubkey + sig

The consensus engine automatically detects signature type by length: 65-byte signatures are processed as ECDSA; 3,732-byte signatures trigger Dilithium verification. Public keys for Dilithium validators are stored in the snapshot's PubKeys map.

6.2 VM-Level PQC Opcodes

The PROBE VM provides native opcodes for post-quantum cryptographic verification:

Opcode	Algorithm	NIST Level	Gas Cost
`falcon512_verify`	Falcon-512	Level 1	800
`ml_dsa_verify`	ML-DSA (Dilithium)	Level 2	800
`slh_dsa_verify`	SLH-DSA (SPHINCS+)	Level 3	1,200
`secp256k1_recover`	secp256k1 ECDSA	Classical	400

Smart contracts can verify any of these signature schemes natively, without precompile workarounds or library overhead.

6.3 Hash Functions

Opcode	Function	Output	Gas Cost
`sha3`	SHA3-256 (Keccak)	32 bytes	200
`shake256`	SHAKE256 (XOF)	32 bytes	200

---

7. The PROBE Language

7.1 Design Philosophy

The PROBE Language is an agent-first programming language designed for AI agents and smart contracts. Its design is guided by five principles:

1. **Resource safety.** Linear types ensure that assets cannot be duplicated or silently lost — inspired by Move's resource semantics.
2. **Agent nativity.** `agent`, `spawn`, `send`, and `recv` are first-class language constructs, not library abstractions.
3. **Token efficiency.** ASCII-only syntax aligned with BPE tokenizers targets ~70 tokens per typical task, optimizing for LLM code generation.
4. **Bytecode verifiability.** Safety properties hold at the bytecode level, even if the compiler is buggy.
5. **Minimal grammar.** Regular or weakly context-free grammar enables zero-overhead constrained decoding for AI code generators.

7.2 Syntax Example

```
agent Echo {
    state {
        count: u64,
    }

    msg handle(data: bytes) -> bytes {
        self.count += 1;
        data
    }
}

resource Token {
    balance: u64,
}

fn transfer(from: &mut Token, to: &mut Token, amount: u64) {
    require(from.balance >= amount, "insufficient balance");
    from.balance -= amount;
    to.balance += amount;
}
```

7.3 Keywords

The language defines 38 keywords organized by function:

Category	Keywords
Control Flow	`fn`, `let`, `mut`, `if`, `else`, `match`, `for`, `in`, `while`, `return`, `break`, `continue`
Type System	`type`, `struct`, `enum`, `trait`, `impl`, `as`, `self`, `resource`
Linear Ops	`move`, `copy`, `drop`
Agent	`agent`, `msg`, `send`, `recv`, `spawn`, `state`
Blockchain	`tx`, `emit`, `require`, `assert`
Visibility	`pub`, `use`, `mod`
Literals	`true`, `false`, `nil`

7.4 Linear Type System

The PROBE Language enforces linear resource safety through its type checker:

Invariants:

- Each linear binding must be consumed exactly once (moved or explicitly dropped).
- Using a binding after it has been moved is a compile-time error (`ErrUseAfterMove`).
- Leaving a linear binding unconsumed at scope exit is a compile-time error (`ErrUnconsumedResource`).
- Calling `drop` on a non-linear value is an error (`ErrDropNonResource`).

The `resource` keyword declares a linearly-typed struct. Resources cannot be implicitly copied — they must be explicitly `move` d to transfer ownership or `drop` ped to destroy them. This prevents entire categories of smart contract bugs: token duplication, accidental token loss, and double-spend vulnerabilities.

7.5 Compiler Pipeline

```
PROBE source → Lexer → Parser → AST → Type Check (linear types)
  → SSA IR → Optimize (SCCP, DCE, CSE) → Verify → Bytecode
```

Stage	Description
Lexer	ASCII-only, single-pass, BPE-aligned tokenizer
Parser	Recursive descent with Pratt expression parsing (11 precedence levels)
AST	22 expression types, 10 statement types, 9 declaration types
Type Checker	Full linear type checking with scope tracking
SSA IR	46 IR opcodes, phi nodes, basic block CFG
Optimizer	Sparse Conditional Constant Propagation (SCCP), Dead Code Elimination (DCE), Common Subexpression Elimination (CSE)
Verifier	Move-inspired bytecode verification — validates safety post-compilation
Codegen	Fixed-width 4-byte instruction encoding

7.6 Standard Library

Module	Description
`agent`	Agent identity, reputation, capability, message types
`chain`	Block, Transaction, State, Log types
`crypto`	SHA3, SHAKE256, Falcon-512, ML-DSA, SLH-DSA verification
`math`	J-style array operations (Map, Zip, Filter, Reduce, Iota, Dot)

---

8. PROBE Virtual Machine

8.1 Architecture

The PROBE VM is a register-based virtual machine optimized for smart contract execution:

Property	Value
Registers	256 general-purpose (R0–R255)
Word Size	64-bit unsigned
R0	Zero register (reads return 0, writes discarded)
Instruction Width	4 bytes (fixed)
Memory	Linear byte-addressable heap, 4 MiB limit
Gas Metering	Per-instruction accounting

8.2 Instruction Encoding

3-Address format: `[opcode:8][a:8][b:8][c:8]`

Wide-immediate format: `[opcode:8][a:8][imm_hi:8][imm_lo:8]` where `imm16 = (imm_hi << 8) | imm_lo`

8.3 Opcode Set

The PROBE VM defines 59 opcodes across 12 categories:

Arithmetic (6 opcodes)

Opcode	Hex	Operation
`add`	0x00	R[a] = R[b] + R[c]
`sub`	0x01	R[a] = R[b] - R[c]
`mul`	0x02	R[a] = R[b] × R[c]
`div`	0x03	R[a] = R[b] / R[c]
`mod`	0x04	R[a] = R[b] % R[c]
`neg`	0x05	R[a] = -R[b]

Bitwise (6 opcodes)

Opcode	Hex	Operation
`and`	0x06	R[a] = R[b] & R[c]
`or`	0x07	R[a] = R[b] | R[c]
`xor`	0x08	R[a] = R[b] ^ R[c]
`not`	0x09	R[a] = ~R[b]
`shl`	0x0A	R[a] = R[b] << (R[c] & 63)
`shr`	0x0B	R[a] = R[b] >> (R[c] & 63)

Comparison (6 opcodes)

Opcode	Hex	Operation
`eq`	0x0C	R[a] = (R[b] == R[c]) ? 1 : 0
`neq`	0x0D	R[a] = (R[b] != R[c]) ? 1 : 0
`lt`	0x0E	R[a] = (R[b] < R[c]) ? 1 : 0
`lte`	0x0F	R[a] = (R[b] <= R[c]) ? 1 : 0
`gt`	0x10	R[a] = (R[b] > R[c]) ? 1 : 0
`gte`	0x11	R[a] = (R[b] >= R[c]) ? 1 : 0

Load/Store (6 opcodes)

Opcode	Hex	Operation
`load_const`	0x12	R[a] = Constants[imm16]
`load_true`	0x13	R[a] = 1
`load_false`	0x14	R[a] = 0
`load_nil`	0x15	R[a] = 0
`move`	0x16	R[a] = R[b]; R[b] = 0 (linear semantics)
`copy`	0x17	R[a] = R[b]

Memory (4 opcodes)

Opcode	Hex	Operation
`load_mem`	0x18	R[a] = Memory[R[b] + offset]
`store_mem`	0x19	Memory[R[a] + offset] = R[b]
`alloc`	0x1A	R[a] = alloc(R[b] bytes)
`free`	0x1B	free(R[a])

Control Flow (6 opcodes)

Opcode	Hex	Operation
`jump`	0x1C	PC = imm16 × 4
`jump_if`	0x1D	if R[a] != 0: PC = imm16 × 4
`jump_if_not`	0x1E	if R[a] == 0: PC = imm16 × 4
`call`	0x1F	Call function at imm16
`return`	0x20	Return R[a]
`halt`	0x21	Halt with exit code R[a]

Stack Frame (2 opcodes)

Opcode	Hex	Operation
`push`	0x22	Push R[a] onto value stack
`pop`	0x23	Pop into R[a]

Agent (4 opcodes)

Opcode	Hex	Operation
`spawn`	0x24	R[a] = spawn_agent(R[b])
`send`	0x25	send(R[a], R[b])
`recv`	0x26	R[a] = recv()
`self`	0x27	R[a] = current_agent_id

Blockchain (6 opcodes)

Opcode	Hex	Operation
`balance`	0x28	R[a] = balance(R[b])
`transfer`	0x29	transfer(R[a], R[b], R[c])
`emit`	0x2A	emit_event(R[a])
`caller`	0x2B	R[a] = caller_address
`block_num`	0x2C	R[a] = block_number
`block_time`	0x2D	R[a] = block_timestamp

Cryptography (6 opcodes)

Opcode	Hex	Operation
`sha3`	0x2E	SHA3-256 hash
`shake256`	0x2F	SHAKE256 XOF
`falcon512_verify`	0x30	Falcon-512 signature verify
`ml_dsa_verify`	0x31	ML-DSA (Dilithium) verify
`slh_dsa_verify`	0x32	SLH-DSA (SPHINCS+) verify
`secp256k1_recover`	0x33	ECDSA public key recovery

Resource Management (3 opcodes)

Opcode	Hex	Operation
`resource_new`	0x34	Create new resource handle
`resource_drop`	0x35	Destroy resource handle
`resource_check`	0x36	Check if resource is live

Array (4 opcodes)

Opcode	Hex	Operation
`array_new`	0x37	Create new array
`array_get`	0x38	Read array element
`array_set`	0x39	Write array element
`array_len`	0x3A	Get array length

8.4 Gas Schedule

Category	Gas Cost	Examples
Trivial	1	move, load immediate, compare
Arithmetic	3	add, sub, neg
Multiply	5	mul
Divide/Mod	10	div, mod
Bitwise	2	and, or, xor, not, shl, shr
Memory	5	load, store, alloc, free
Jump	3	any branch instruction
Call	20	function call overhead
Crypto (base)	200	sha3, shake256
Falcon-512 Verify	800	4× crypto base
ML-DSA Verify	800	4× crypto base
SLH-DSA Verify	1,200	6× crypto base
Secp256k1 Recover	400	2× crypto base
Agent	50	spawn, send, recv
Blockchain	30	balance, transfer, emit

8.5 Contract Format

PROBE bytecode is stored on-chain with a magic prefix for identification:

```
[0x50 0x52 0x42 0x45]  "PRBE" magic prefix (4 bytes)
[num_constants]          Little-endian uint32 (4 bytes)
[constants...]           num_constants × 8 bytes (uint64 pool)
[bytecode...]            Remaining bytes (4-byte aligned instructions)
```

The VM validates the magic prefix before execution and rejects non-PROBE code.

8.6 Error Handling

Error	Cause
`ErrOutOfGas`	Gas budget exceeded
`ErrDivisionByZero`	Division or modulo by zero
`ErrInvalidOpcode`	Unrecognized opcode byte
`ErrStackUnderflow`	Pop on empty stack
`ErrResourceFault`	Double-drop or use-after-move
`ErrNilReceive`	Recv with no pending message

---

9. Network Protocol

9.1 Protocol Identity

Property	Value
Protocol Name	`probe`
Primary Version	ETH66 (19 message types)
Legacy Version	ETH65 (17 message types)
Max Message Size	10 MB

9.2 Message Types

Code	Message	Direction	Description
0x00	Status	Bidirectional	Handshake with chain state
0x01	NewBlockHashes	Broadcast	Announce new block hashes
0x02	Transactions	Broadcast	Propagate transactions
0x03	GetBlockHeaders	Request	Request header range
0x04	BlockHeaders	Response	Return headers
0x05	GetBlockBodies	Request	Request block bodies
0x06	BlockBodies	Response	Return bodies
0x07	NewBlock	Broadcast	Propagate full block
0x08	NewPooledTxHashes	Broadcast	Announce pending tx hashes
0x09	GetPooledTxs	Request	Request pending transactions
0x0A	PooledTxs	Response	Return pending transactions
0x0D	GetNodeData	Request	Request state trie nodes
0x0E	NodeData	Response	Return trie nodes
0x0F	GetReceipts	Request	Request transaction receipts
0x10	Receipts	Response	Return receipts
0x11	BehaviorProof	Broadcast	PoB behavior proof propagation
0x12	Ack	Broadcast	PoB acknowledgment propagation

9.3 Peer Compatibility

Peers must match on:

- Network ID
- Genesis block hash
- Fork ID (compatibility check)

10. Superlight Hybrid DEX

10.1 Overview

Superlight is ProbeChain's native on-chain decentralized exchange, implemented as a dedicated transaction type (Type 4) for optimized processing and reduced gas overhead.

10.2 Configuration

Parameter	Default	Description
Maker Fee	10 bps (0.1%)	Fee for limit order placement
Taker Fee	30 bps (0.3%)	Fee for market order execution
Max Orders/Account	100	Open order limit per account

10.3 Transaction Format

Superlight transactions use the EIP-2718 typed transaction envelope with type byte 4:

```
SuperlightTx {
    ChainID:    uint256,
    Nonce:      uint64,
    GasTipCap:  uint256,     // Max priority fee
    GasFeeCap:  uint256,     // Max total fee
    Gas:        uint64,
    To:         address,     // DEX settlement contract
    Value:      uint256,
    Data:       bytes,       // RLP-encoded DEX operation
    AccessList: AccessList,
    V, R, S:    uint256      // Signature
}
```

10.4 Supported Operations

- **Order Placement** — Submit limit or market orders
- **Order Cancellation** — Cancel open orders
- **Order Settlement** — Execute matched orders
- **Liquidity Provision** — Add liquidity to trading pairs

11. Conclusion

ProbeChain V2.0 represents a comprehensive rethinking of blockchain architecture for the AI agent era. By replacing energy-intensive mining with behavioral scoring, achieving sub-second finality through StellarSpeed, integrating post-quantum cryptography at every layer, and providing a purpose-built programming language for agent-driven smart contracts, ProbeChain establishes a foundation for the next generation of decentralized applications.

The Stellar-Class Resilience features — AtomicTime, Rydberg receivers, and RF block propagation — position ProbeChain uniquely for scenarios ranging from disaster-resilient infrastructure to interplanetary communication networks.

Key Differentiators

Feature	ProbeChain V2.0	Typical L1
Consensus	Proof-of-Behavior (5D scoring)	PoS / PoW
Block Time	400 ms	2–15 seconds
Quantum Safety	Native Dilithium + VM opcodes	None / roadmap
Smart Contract VM	Register-based, 64-bit, linear types	Stack-based, 256-bit
Agent Support	First-class language primitives	Library-level
Off-Grid Consensus	Stellar RF layer (HF–THz)	Internet-dependent
Time Precision	Nanosecond (AtomicTime)	Second-level

References

- NIST FIPS 204: Module-Lattice-Based Digital Signature Standard (ML-DSA / Dilithium)
- NIST FIPS 205: Stateless Hash-Based Digital Signature Standard (SLH-DSA / SPHINCS+)
- NIST FIPS 206: FFT-over-NTRU-Lattice-Based Digital Signature Standard (Falcon)
- IEEE 1588-2019: Precision Time Protocol (PTP)
- ITU-T G.811: Timing characteristics of primary reference clocks